APPLICATIONS OF NUMERICAL OPTIMIZATION IN GRAPHICS AND GAMES

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Albert Julius Liu

May 2018

APPLICATIONS OF NUMERICAL OPTIMIZATION IN GRAPHICS AND GAMES

Albert Julius Liu, Ph.D.

Cornell University 2018

This dissertation begins with a camera calibration system, CALIBER, which solves pose estimation problems consisting of two types of constraints: relative pose constraints, resulting from measurements, such as found in SLAM and motion estimation problems; and rigidity constraints, the notion of objects that are rigidly attached to each other so that their relative pose is fixed over time even if that pose is not known a priori. We show that this problem is NP-hard, but demonstrate an algorithm that works well in practice. Applications include calibrating goniometers for accurate measurement of light reflection.

We then proceed further down the pipeline from measurement of light reflection to the modeling of the appearance of surfaces. Specifically, we examine the appearance of wood. While suitable BRDF models exist, the texture parameter maps for these wood BRDFs are difficult to author—good results have been shown with elaborate measurements for small flat samples, but these models are not much used in practice. Furthermore, mapping 2D image textures onto 3D objects leads to distortion and inconsistencies. Procedural volumetric textures solve these geometric problems, but existing methods produce much lower quality than image textures. This chapter aims to bring the best of all these techniques together: we present a comprehensive volumetric simulation of wood appearance, including growth rings, color variation, pores, rays, and growth distortions. The fiber directions required for anisotropic specular figure follow naturally from the distortions.

The final piece of the dissertation is another application of numerical optimization, this time in games, both in the sense of entertainment and in the sense of game theory. A key challenge in game design is achieving balance between the strategies available to the players. We model the balancing problem as modifying a zero-sum game, using one variable per strategy, so that every strategy has an incentive to be employed. We begin with a special case where these variables affect player payoffs multiplicatively, and show that the simple Sinkhorn-Knopp algorithm can be used to balance the game. We then proceed to analyze the more general case where the variables have a monotonic effect on payoffs, and show that it is amenable to standard optimization methods.

BIOGRAPHICAL SKETCH

Albert Julius Liu completed bachelor's degrees in Electrical Engineering and Computer Science (EECS) and Physics at the University of California, Berkeley. He is now completing his doctorate at Cornell University in Computer Science. He studied computer graphics for several years, but has taken a new interest in game theory.

I would like to thank the following people:

- My advisor Steve Marschner, for his patience and willingness to entertain my shifts in interests, even far afield,
- Taryn Mattice, my surest friend during my time in Ithaca,
- And most of all my parents, the most steadfast force in my life.

ACKNOWLEDGEMENTS

I received a Jeff Hawkins and Janet Strauss Fellowship, which generously provided funding for my first year at Cornell.

"Caliber: camera localization and calibration using rigidity constraints": We would like to thank Wenzel Jakob, Pramook Khungurn, Bruce Walter, and Rundong Wu for testing out Caliber on real problems and providing feedback. Funding for this work was provided by National Science Foundation grant IIS-1011919, by the Intel Science and Technology Center for Visual Computing, and by a gift from Autodesk.

"Simulating the structure and texture of solid wood": We would like to thank Roberto Ziche for suggesting wood model features and creating parameter presets, John Hutchinson for support and suggestions, Kate Salesin and Lydia Wang for their help in preparing and photographing the wood samples, Karl J. Niklas for his help in understanding wood structure, and François Guimbretière for providing lab space for preparing the wood samples. Funding for this work was provided by National Science Foundation grant IIS-1011919 and by a gift from Autodesk.

"Balancing zero-sum games with one variable per strategy": We would like to thank Erik Andersen, Jimmy Briggs, Shuo Chen, and Éva Tardos for their aid in preparing this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

I began my PhD in the fall of 2010, and started working with Steve Marschner in mid-2011.

1.1 Caliber

Our graphics lab has a device called the "spherical gantry": a robot arm that can be programatically positioned at any angle around a target pedestal. We used it to measure the reflection of light from various surfaces. However, achieving accurate results required accurate calibration of the position and angle of the target object with respect to the robot arm. Previously we had done this using special-purpose code, but we were interested in whether this calibration problem could be generalized: useful for adding additional components to the gantry setup, or for entirely different setups. The Caliber system grew out of this initial inspiration, the key insight being that knowledge of rigidity in the system (e.g. that of a rigid robot arm segment) is not only a useful quantity to determine (e.g. finding the length of the arm), but the very existence of that rigidity is itself a crucial piece of information in calibration problems.

1.2 Wood

After finishing work on Caliber I moved further down the pipeline. One of the materials we were interested in modeling was wood. My advisor had developed

a BRDF model of the reflectance of light from wood back in 2005 [55], but the difficulty of acquiring data for the model remained an obstacle—the original paper used the spherical gantry to measure specific pieces of wood, but there are not many such devices in the world. To create a more accessible system I decided to develop a procedural, solid wood texture. I approached this by studying the microscopic structure of wood, observing which such structures corresponded to distinctive visual features of wood, and then modeling such structures in the procedural texture. This project was supported by Autodesk, who implemented this texture into one of their products, Fusion 360. This was instrumental in generating renderings for publication.

1.3 Zero-sum games

This was a problem I had been thinking about in some form or another since high school. The original inspiration was the Vehicle Design Rules for *Warhammer 40,000* [13]: a system for assigning statistics and costs to self-constructed vehicle miniatures (as opposed to the standard store-bought vehicles). I wondered whether there was some way of automatically estimating an appropriate cost for particular statistics based on the mechanics of the game.

Later on, during undergraduate I helped develop some games using the Spring RTS engine [88]. While working on these games, I was similarly interested in automatically estimating an appropriate cost for the various units available in the games. At the time I only got as far as analyzing individual units in isolation using Lanchester's laws. We recognized that a better estimation system would require some consideration of interactions between units. For example, a unit with slightly

longer range than major static defenses can bombard those defenses from safety, making it significantly more powerful than a unit with a slightly shorter range than those defenses, which must enter the field of fire of such defenses in order to attack them.

At these times I did not have the knowledge to solve, or even rigorously define, problems of this type. Over a decade after the initial inspiration I revisited the question and found that game theory provided a way to express some problems of this type; and that numerical optimization, which I had been introduced to in my first project CALIBER, provided a way of solving it. This became the third and final paper of my PhD.

CHAPTER 2

CALIBER: CAMERA LOCALIZATION AND CALIBRATION USING RIGIDITY CONSTRAINTS

2.1 Introduction

Many computer vision applications—including shape and appearance capture, vision for robots and autonomous vehicles, motion tracking for film production, and human-computer interaction systems—require calibration of systems of cameras.

In the simplest case, one is only interested in the intrinsic parameters of cameras such as focal length. This can be achieved by capturing a set of views of a calibration target such as a checkerboard, and solving for independent poses for each view along with camera intrinsics. However, in many other cases, the camera system has additional internal geometric constraints. The simplest example is a stereo pair: the relative pose between the two cameras remains constant, and learning the relative pose is an important goal of the calibration process. At the same time, the rigid link between the cameras provides an additional constraint that can be used to improve the accuracy of the calibration solution. Nonlinear optimization algorithms can optimize over such relative poses, but they require a good initial guess in order to succeed. Although specialized methods have been developed for various specific systems, such methods are limited to calibrating a single type of setup, requiring that a new method be developed whenever a new type arises.

Rigidity constraints and SL-R. In this chapter, we observe that the core geometric problem underlying this kind of calibration can be expressed using **rigidity**

constraints: the notion that two objects can be rigidly attached to each other, so that their relative pose is fixed over time even if that pose is not known *a priori*. Based on this observation, we introduce the sensor localization with rigidity problem (**SL-R**), in which the goal is to find a set of absolute poses that satisfies a set of relative pose constraints and rigidity constraints. We show that SL-R is NP-hard, but give a inference-based algorithm for SL-R that performs well in practice.

Caliber. We then proceed to describe CALIBER, a calibration system built around our SL-R algorithm. CALIBER begins with a kinematic tree, which describes the structure of the calibration setup and the measurements taken. The kinematic tree consists of a set of nodes, representing objects, and a set of observations that describe the relative poses and rigidity constraints between the objects when measurements were taken. For cameras, these relative pose estimates as well as initial intrinsic estimates can come from e.g. the method of [93].

CALIBER takes the kinematic tree and uses our SL-R algorithm to generate an estimate for the rigidity constraints. Finally, it performs nonlinear optimization over the extrinsics and intrinsics to produce a locally least-squares optimal solution to the calibration problem in terms of reprojection error. The use of rigidity constraints allows CALIBER to deliver greater generality, usability, and accuracy.

Generality. Previous methods either only support relative pose estimates and not rigidity constraints, or are specialized for a particular type of calibration setup and require the user to develop a new method for solving any different or novel case.

In contrast, CALIBER is able to handle these rigidity constraints, and only

requires a description of the calibration setup in order to solve the problem. This relieves the user from the responsibility of working out a procedure for solving each new calibration problem.

Usability. Rigidity constraints also allow CALIBER to take input in a form that is easy for the user to express, and produce output in a form that the user is directly interested in. We present the user with a representation called a **kinematic tree**, which can be thought of as a scene graph, as a transformation hierarchy, or as a generalization of the kinematic chains used to describe robot arms. This representation is equivalent in power to SL-R, but provides a simpler and more intuitive input format to the user. In turn, this makes it easier for the user both to describe a calibration setup and to interpret the solution produced by CALIBER. Furthermore, the kinematic tree defines a specific, unambiguous, and minimal parameterization of the relative poses of the calibration setup, which is important for our optimization stage.

Accuracy. Finally, rigidity constraints allow CALIBER to use a minimal set of relative pose parameters, instead of having a separate absolute pose for every measurement. This avoids overfitting, making the underlying optimization problem fundamentally better conditioned, which results in more accurate calibration results.

Evaluation. We demonstrate CALIBER on a variety of multi-camera setups, from established problems like multiview stereo rigs to new problems, such as self-calibration of the relative pose of the LCD display and the camera on a mobile device. We also discuss how the calibration results can be evaluated to learn about

the numerical strength of a calibration setup.

Source code for CALIBER is publicly available at `http://www.cs.cornell.edu/projects/caliber`.

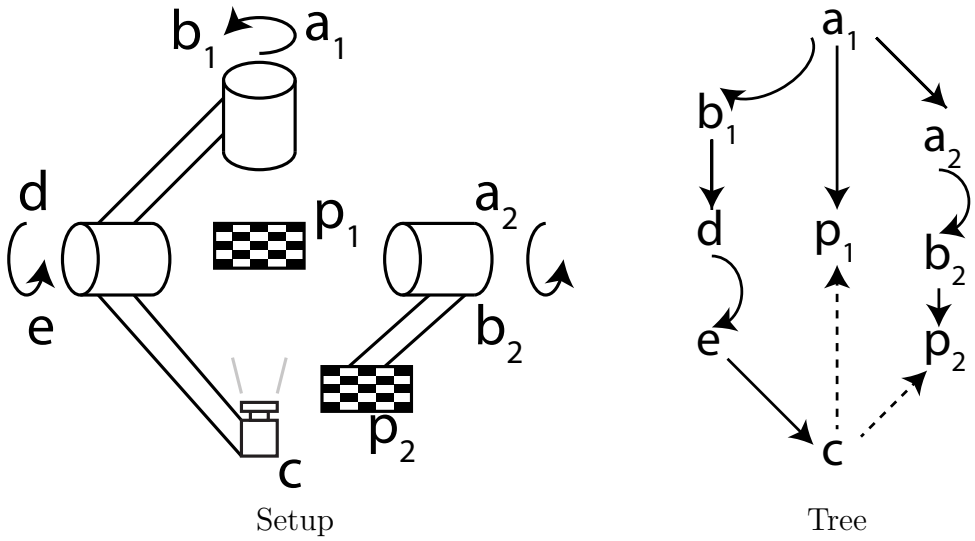


Figure 2.1: Representation of a two-arm spherical gantry. The second arm has only a single axis of rotation. See Figure 2.4 for more examples.

2.2 Related work

2.2.1 Camera calibration

The most widely used camera calibration methods in machine vision are descended from Tsai's work on practical calibration from views of 2D or 3D sets of known points [85]. Later Zhang [93] combined these ideas with self-calibration [56] to produce a method that fits n camera poses and one set of camera intrinsics to n images of a planar target. This approach has been widely adopted, in part

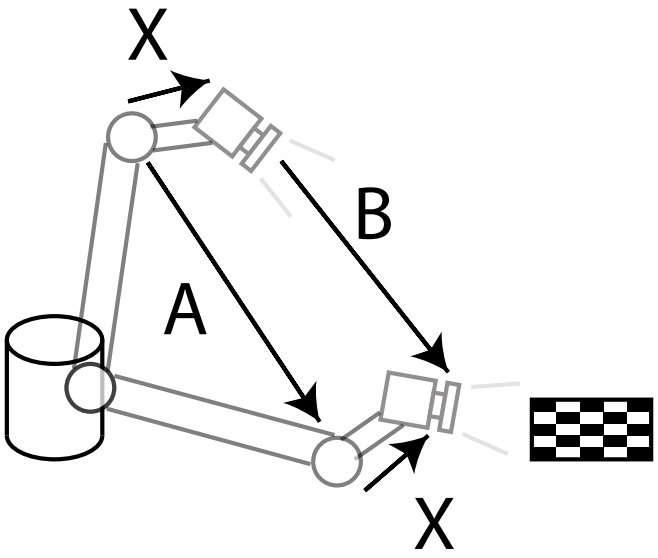


Figure 2.2: The $AX = XB$ problem induced by the hand-eye calibration problem. In this case, A is the movement of the hand between two exposures (measured using the arm's encoders), B is the movement of the camera between the same two exposures (measured using the camera's views of the calibration target), and X is the unknown we wish to find, namely the relative pose between the hand and camera.

because of the convenience of planar targets and in part due to Bouguet's freely available implementation [6]. The problem solved by Zhang's method corresponds in Caliber to a tree with two nodes, a camera and a calibration target. While this method produces accurate camera intrinsics, the pose estimates can be noisy, since each can only draw upon a single view.

2.2.2 Sensor localization

The problem of determining a set of absolute poses that best fits a set of relative pose estimates has been extensively studied in various contexts. These can be

represented as graphs where each vertex is an absolute pose, and edges represent relative pose estimates. In simultaneous localization and mapping (SLAM) (e.g. [84]), vertices correspond to positions of a robot over time and landmarks observed by the robot, while edges correspond to odometry measurements and relative pose estimates generated by the robot's views of those landmarks. Since the composition of rigid transformations around a cycle should equal the identity, cycles in the graph can lead to a more accurate solution [22].

Similarly, structure-from-motion methods (e.g. [92]) use graphs and loop-closing to determine their alignments, and Govindu [25, 26] proposes an efficient method of combining the up to $\frac{n(n-1)}{2}$ possible relative pose estimates from n views.

However, while these methods are general in terms of problems involving relative pose estimates, they do not handle rigidity constraints. [41] simultaneously produces a SLAM solution and calibrates the sensors on the robot including their relative positions, similar to the optimization step in Caliber. However, they take as given a good initial guess for the calibration parameters and do not face the initialization problem represented by SL-R.

2.2.3 Rigidity constraints and $AX = XB$

Beyond localization problems involving only relative pose estimates, there are existing special-purpose methods that use solutions to a system of rigid transformation equations of the form $AX = XB$ to localize and calibrate camera systems. The oldest and best-known application of this is robot hand-eye calibration, in which the pose (relative to the hand) of a camera carried by a robot arm is established using views of a fixed calibration target. In this case, A is the movement of the

hand between two exposures (measured using the arm's encoders), B is the movement of the camera between the same two exposures (measured using the camera's views of the calibration target), and X is the unknown we wish to find, namely the relative pose between the hand and camera. The robot hand-eye calibration scenario is depicted in Figure 2.2. This was introduced simultaneously by [71] and [86]. This problem has been an active research area for some time; see [81] for a more complete survey. The same $AX = XB$ solvers have been used more recently [21] to find the relative poses of groups of rigidly connected cameras that look in different directions, such as cameras mounted on an autonomous vehicle. Even though there is no actuator in the problem, it reduces to the same $AX = XB$ problem: here A is the movement of one camera between two exposures, and B is the movement of another camera between the same two exposures.

The key common feature in these calibration problems, as well as others such the multiply-jointed robot arm of our spherical gantry setup shown in Figure 2.4 (e), or the facing smartphones setup (f), is rigidity. Two objects, such as cameras or robot joints, can be rigidly attached to each other. The result is that the relative pose between them remains the same as the objects move, regardless of whether the value of that relative pose is initially known. By using rigidity constraints, our algorithm subsumes these previous methods, and generalizes them so that novel systems can be calibrated, including more complex systems such as our spherical gantry that may involve solving more than one such system of equations.

Mathematically, rigidity constraints allow the same rigid transformation variable to appear multiple times in systems of $AX = XB$ equations. Our algorithm leverages existing $AX = XB$ solutions as part of its inference process. In the unambiguous case we use Park and Martin's [61] closed-form solution which produces

a least-squares optimal solution. However, it is possible for a system of $AX = XB$ equations to have an ambiguous solution; that is, the solution space for such a system may have one or more dimensions. [15] gives an analysis of the possible cases based on screw theory, and we use a set of methods based on this in the case of an ambiguous system of $AX = XB$ equations. These methods are discussed in Appendix 2.7. Our algorithm automatically determines, selects, and solves these systems of equations based on a set of heuristics, thus freeing the user from having to work out a procedure for solving a particular instance.

2.2.4 Complexity

We show that SL-R, as well as its uniqueness variant, are NP-hard. Similar results were proved for structure-from-motion with missing data [58] and robot localization [19], the latter reducing to the same (unique) partition problem as our proof. However, both the problem and structure of our proof are distinct from these.

2.3 Sensor localization with rigidity (SL-R)

Caliber ultimately computes its calibration result using a nonlinear optimization to jointly determine the internal parameters of all cameras and all unknown pose relationships in the system to be calibrated. For this nonlinear optimization to succeed, it is critical to provide a good initial starting point: an estimated solution that is approximately consistent with all the available data. Internal parameters can be estimated, one camera at a time, using existing techniques, but in many cases estimating poses requires reasoning globally about the geometry of the sys-

tem. This section describes one of the key innovations of this chapter: a method of finding consistent pose estimates in systems that involve rigidity constraints as well as relative pose constraints.

Abstracted away from any particular calibration setup or type of sensor, the problem boils down to approximately solving a system of constraints on a set of poses. Each pose represents the position of an element of the system, such as a camera or calibration target, at one point in time relative to some other part of the system. The constraints come in two types: **relative pose constraints** come from measurements relating two elements of the system at a particular time, and **rigidity constraints** express the fact that certain relationships remain fixed for all time. (See Section 2.4.4 for details of how a calibration problem is reduced to a set of constraints.)

In this section we introduce this problem, which we call sensor localization with rigidity (SL-R), and propose an algorithm that solves many instances of it. We also show in Appendix 2.8 that the general problem is NP-hard.

2.3.1 SL-R definition

SL-R is a constraint-satisfaction-like problem: given a set of relative pose constraints (each of which fixes the relative pose between pairs of absolute poses) and a set of rigidity constraints (each of which forces a set of relative poses to be the same), determine a set of absolute poses that fit these constraints. SL-R can be

defined symbolically as follows: Subject to contraints,

for all

$$i \in \{1 \dots n\} \qquad \text{(absolute pose indices)}$$

find

$$P_i \in SE(3) \qquad \text{(absolute poses)} \tag{2.1}$$

fixing $P_0 = I$ as the global frame.

Constraints come in two types. First, we have relative pose constraints that force a relative pose to take a given value T. Each such constraint has the form

given

$$(i, j) \in \{1 \dots n\}^2 \qquad \text{(relative pose index pair)}$$

$$T \in SE(3)$$

require

$$P_{ij} = P_i^{-1} P_j = T \tag{2.2}$$

There may be any number of such constraints.

Second, we have rigidity constraints that force all relative poses of a set L to take the same value as each other. Each such constraint has the form

given

$$L \subset \{1 \dots n\}^2$$

require

$$\forall (i, j) \in L, \forall (k, \ell) \in L \qquad P_{ij} = P_{k\ell} \tag{2.3}$$

Again, there may be any number of such constraints.

With only relative pose constraints, this problem is similar to that faced by SLAM and other localization problems. But the addition of rigidity constraints introduces considerably more complexity, in fact making the problem NP-hard even in the exact case as we show in Appendix 2.8.

These equations may be underdetermined; depending on the set of constraints it may not be possible to determine a unique solution (see Section 2.3.2). Indeed, the underdetermined space is rich enough that the problem of determining whether a unique solution exists is *also* NP-hard. Intuitively, we can build a gadget that represents a relative pose subproblem involving a two-fold ambiguity. This forces a choice between two options; by using rigidity constraints we can enforce that choice across the entire problem. From here it is a matter of geometrically encoding a NP-complete problem on such choices.

In practice, these equations will virtually always be overdetermined. Ideally, any instance of SL-R arising from a real scenario would have at least one solution: the actual real-world poses of the objects. However, since real-world measurements are imperfect, it is not generally possible to determine a set of absolute poses that perfectly fits a given set of measurements. We must produce a reasonable approximate solution even in the presence of noise.

We now proceed to explain the components of SL-R in more detail.

Poses. The variables of SL-R are an indexed set of n absolute poses P_i (fixing P_0 as the "root" pose representing the global coordinate frame).

Each absolute pose is a rigid transformation representing an object (such as a camera or robot joint) *at one point in time*, as given in Equation 2.1. By "one point

in time" we are referring to the poses of the objects in the system at the moment a measurement is taken. This means that a single object could correspond to multiple different poses at different points in time; that is, when different measurements are taken. Section 2.4.4 will explain how these multiple poses are generated from a node in the kinematic tree.

A useful alternative expression of the poses is to consider the n^2 relative poses $P_{ij} = P_i^{-1} P_j$ between every ordered pair i, j of absolute poses, as is done in e.g. [84]. These relative poses represent the pose of one object relative to another at one point in time. From this definition, the composition of relative poses around any cycle must equal the identity. This expression has the advantage of allowing us to consider each cycle to be an equation.

Relative pose constraints. Sensors such as cameras or actuator encoders can measure the relative pose between two objects. For example, cameras can measure the pose between themselves and a calibration target, whereas the value of an actuator encoder implies a particular relative pose induced by the joint. To represent these, we have relative pose constraints, which give the relative pose between two absolute poses, as given in Equation 2.2.

Rigidity constraints. In addition to these relative pose constraints, we have rigidity constraints. A rigidity constraint is defined by a set of relative poses. It constrains these relative poses to be the same, even if that rigid transformation is not known *a priori*, as given in Equation 2.3. We consider these relative poses to share a **label** and associate a common **label transformation** with that label.

Rigidity constraints represent the notion that two objects can be rigidly at-

tached to each other. Though their absolute poses may change from measurement to measurement, the relative pose between them remains constant. Examples of this include the cameras of a rigid camera cluster, two joints at opposite ends of a rigid link, and the integrated camera of a smartphone and its screen.

Solution definition. In the exact case, the solution to an instance of SL-R assigns a rigid transformation to each absolute pose P_i (or equivalently to each relative pose P_{ij}, with the constraint that the composition of relative poses around any cycle must equal the identity) such that all of the constraints are satisfied. In practice, the relative pose constraints will come from noisy measurements, and so it will not be possible to satisfy all constraints exactly. To account for this, we use local averaging to combine multiple inferences of a pose when more than one is available. Then, in the end, we perform nonlinear optimization afterwards to refine the result to a locally least-squares optimal solution in terms of reprojection error; this is described in Section 2.4.

Algorithm 1: SL-R algorithm

```
P = 4n × 4n identity matrix
foreach relative pose constraint do
    fill in the corresponding block of P with the constraint transformation
    fill in the transpose block with the inverse
while not completely solved do
    if at least one available triangle closing exists then
        perform triangle closing (direct rule)
    else if newly solved label exists then
        propagate label transformation to relative poses with that label
         (label rule)
    else
        foreach label do
            prospectively run AX = XB solver on all known relative poses
             with a label
        pick the most determined label and use that value for the label
         (AX = XB rule)
```

Properties of As and Bs	Ambiguities
At least two non-parallel rotations	Completely determined
Parallel rotations	Ambiguous translation along rotation axis
Any rotation	Ambiguous rotation and translation along rotation axis
No rotations, at least two non-parallel translations only	Ambiguous translation
No rotations, any translation	Ambiguous rotation along translation axis, ambiguous translation
No movement or no information	Completely ambiguous

Table 2.1: Classification of ambiguity of $AX = XB$ equations.

2.3.2 SL-R algorithm

In this section we present an algorithm for SL-R. Our algorithm runs in polynomial time but is not guaranteed to produce a correct solution; as we will show in Appendix 2.8, SL-R is NP-hard and so no known polynomial-time algorithm can produce a correct solution in all cases. However, as part of Caliber, our algorithm has worked in all practical solvable cases we have encountered so far, and is additionally able to detect and classify common forms of ambiguity.

General strategy and considerations

Recall from Section 2.3.1 that a solution to SL-R can be defined as an assignment of relative poses P_{ij} such that the composition of relative poses around any cycle is the identity, and all constraints are satisfied. This means that each cycle of poses corresponds exactly to an equation of rigid transformation matrices, with these equations being related to each other via the constraints. Our algorithm uses P_{ij} as the variables.

There are some special considerations that arise:

- Real measurements are noisy. If there are multiple ways of estimating a particular relative pose, each may produce a different result. Therefore, a way to average these estimates can improve the estimate.
- Even in the absence of noise, many instances of SL-R are inherently ambiguous; that is, there exists more than one solution. Fortunately, it is usually practically sufficient to select *a* solution in this case: if more than one solution fits the calibration data, this usually implies that they will make identical predictions as well. It is also useful to automatically detect and, where possible, classify these ambiguities.

Our algorithm incrementally builds a solution by applying a series of inference rules. Each of these rules produces an estimate of a relative pose by solving an available equation or system of equations. In order of highest to lowest priority, these are the direct rule, the label rule, and the $AX = XB$ rule. The algorithm is summarized in Algorithm 1.

Initial state

We represent the relative poses of SL-R as a block matrix P whose blocks are the relative poses P_{ij} represented as 4x4 rigid transformation matrices.

The condition that the composition of relative poses around any cycle must equal the identity has two basic implications: $P_{ii} = I$ (each absolute pose is identical to itself), and $P_{ij} = P_{ji}^{-1}$ (the relative pose from one absolute pose to another is the inverse of the relative pose in the other direction).

Therefore, we initialize P to contain I on its diagonal blocks and write the relative pose constraints into the matrix directly. Any block without a relative pose constraint is initially set to 0, indicating an unknown relative pose.

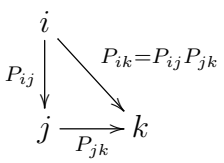
Here is an example with four absolute poses and a single relative pose constraint $P_{2,4} = T$:

$$P = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & T \\ 0 & 0 & I & 0 \\ 0 & T^{-1} & 0 & I \end{bmatrix} \tag{2.4}$$

Direct rule

The direct rule is derived from the constraint that the composition of transformations around any cycle should equal the identity. The simplest case is triangle-closing: if we know the values of two relative poses that both involve the same absolute pose, we can estimate the relative pose between the other two absolute poses as the composition of those two transformations.

A pictorial representation of this triangle-closing, where points represent absolute poses and arrows represent transformations:



(2.5)

The relative pose that closes a longer cycle can be estimated by applying this triangle-closing multiple times. Therefore, at the basic level, our direct rule performs this triangle-closing. A simple method of expressing triangle closing over all possible triples of absolute poses simultaneously is to use block matrix multiplication. Specifically, the block elements of the product $PP = P^2$ are:

$$\left(P^2\right)_{ik} = \sum_{j=1}^{n} P_{ij} P_{jk} \tag{2.6}$$

Each term in the sum is an estimate for the relative pose P_{ik} between absolute poses i and k iff both P_{ij} and P_{jk} are nonzero (that is, known), and zero otherwise. If the sum is zero, the relative pose could not be solved for via triangle closing and is therefore unknown. Otherwise, dividing by the homogeneous coordinate (bottom-rightmost element) gives us an average of several estimates, which we then project onto $SE(3)$ via SVD.

We apply the direct rule repeatedly until no new relative poses are estimated.

Label rule

If the direct rule is unable to make progress, we move on to the label rule, which is a direct application of the rigidity constraints. For each label, if at least one of the relative poses with that label has a known rigid transformation, we set all relative poses with that label to that rigid transformation. Like the direct rule, if there are multiple estimates for the same label, we take the average and project onto $SE(3)$.

AX = XB rule

If neither of the previous rules can be applied, we use our final rule, which relies on solving systems of $AX = XB$ equations. It is this rule which allows our algorithm to solve novel and non-trivial problems.

For every pair $P_{ij}, P_{k\ell}$ of relative poses with the same unknown label X, we check if the relative poses $A = P_{ik}$ and $B = P_{j\ell}$ are both known. If so, the four relative poses form a cycle corresponding to an $AX = XB$ equation with A and B known and X unknown. Pictorially, such an equation looks like this:

$$\begin{array}{ccc} i & \xrightarrow{X=P_{ij}} & j \\ {\scriptstyle A=P_{ik}}\downarrow & & \downarrow{\scriptstyle B=P_{j\ell}} \\ k & \xrightarrow[X=P_{k\ell}]{} & \ell \end{array} \tag{2.7}$$

Since labels can potentially appear many times in SL-R, there may be multiple such cycles for the same label X, each of which produces a different $AX = XB$ equation. Depending on the values of the As and Bs and the number of equations, the solution space for each label may have a different number of dimensions. In other words, some labels may be more constrained by their $AX = XB$ equations than others. We choose the most determined label to solve; if the solution is incompletely determined, we choose the solution that is closest to zero rotation angle and zero translation. In the presence of noise, we use a least-squares solution that accounts for all available $AX = XB$ equations for each label, and when determining how many dimensions the solution space of a system of $AX = XB$ equations has, we have minimum tolerances for which we consider two vectors to be nonzero (translation) or nonparallel (rotation). We report the type of ambiguity of the

chosen label, then return to the previous rules.

The specific cases of $AX = XB$ are discussed in Appendix 2.7. A brief summary appears in Table 2.1.

Possible outcomes

Termination occurs in polynomial time: each outer loop iteration takes polynomial time, and since we can always guess a relative pose in the worst case, there are at most a polynomial number of iterations.

Fully determined solution. If our algorithm reports that the solution is completely determined, the solution is guaranteed to be unique and correct in the exact case. Practically, in the presence of noise, this produces a solution near the unique optimum.

Ambiguity. For some problems there may not exist a unique solution at all. However, this usually does not preclude a useful calibration. For example, a revolute joint (such as in our two-arm case—see Figure 2.4) produces rotations along only a single axis. In this case it is impossible to distinguish (from e.g. observations of a calibration target mounted on the joint) a difference in the location of the joint along the axis from a difference in the distance from the joint to the target along that axis. In this case, our algorithm would set one to zero and solve for the other. In this and all other practical cases we have encountered so far, our algorithm will succeed in finding *a* correct solution via the $AX = XB$ rule, and the arbitrary choices of our algorithm have no effect on the predictions made by the resulting model as long as the predictions do not exercise any degrees of

freedom that were not present in the calibration data. Intuitively, an ambiguity may correspond to a parameter of the system that does not produce a visible effect in the observations. However, as long as future predictions are made under similar conditions, an arbitrary choice made to resolve that ambiguity will not be visible in those predictions either.

Complexity

As discussed in Appendix 2.8 it is possible to construct calibration instances that cannot be resolved by our $AX = XB$ analysis—indeed, that encode NP-complete problems. Given such an instance, our algorithm will likely guess a wrong value for a label. Again, we have not yet come across any practical calibration schemes where this occurs.

2.4 Caliber

The previous section discussed the core challenges in creating a general calibration system that solves the kinds of problems sketched in the introduction and proposed an algorithm for doing so. In this section we proceed to discuss Caliber, a calibration system for geometrically constrained camera systems that is built around that algorithm. By being able to handle rigidity constraints, Caliber provides greater generality than previously existing systems, and by using a final nonlinear optimization stage over all parameters, Caliber produces locally least-squares optimal solutions.

In order to make the system easy to use, the input to Caliber is in the form

of a description of the system being calibrated together with the observations that were made on it. This is equivalent to an SL-R problem instance, but much easier for a user to specify; the system takes care of converting it to an SL-R problem internally.

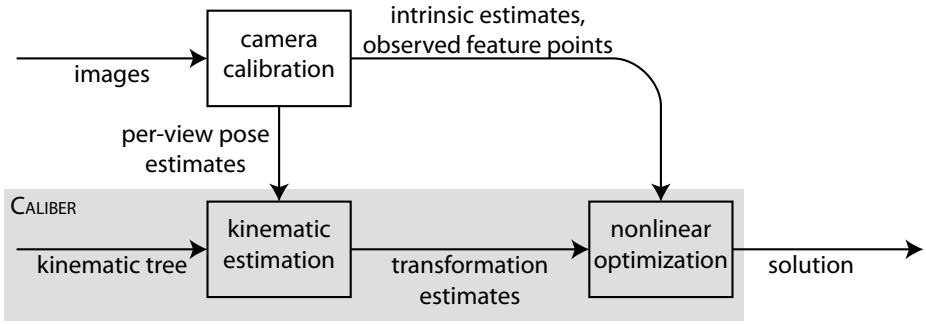


Figure 2.3: The structure of CALIBER: before CALIBER is run, the images are processed by a standard **camera calibration** tool [6] to find image-space features, camera intrinsics, and per-image pose estimates. Next, our **kinematic estimation** phase computes approximate transformations according to a tree-structured model of the setup; finally, a **nonlinear optimization** refines this estimate to a locally least-squares optimal solution.

2.4.1 Overall structure

As illustrated in Figure 2.3, calibration using CALIBER begins with a set of photographs of calibration targets from the camera(s) of the system. Both targets and cameras are attached to the system, and during calibration multiple photographs are taken as the system is taken through its range of motion. We then run Zhang's algorithm to determine the intrinsics of each camera and an estimated pose relating the camera to a calibration object in each view. These poses, together with a kinematic tree describing the geometric constraints, are used in the second stage to estimate the underlying properties of the constrained system accounting for constraints such as rigid links and known motions. Finally, all the system parame-

ters including camera intrinsics are jointly optimized to find a locally least-squares optimal solution.

Camera calibration. The physical calibration process consists of taking photographs from several views from camera(s) of calibration target(s), ideally getting a variety of angles relative to the calibration target(s) and exercising all of the degrees of freedom of the system. Given a set of views, the Zhang method is able to estimate the relative pose between the camera and target for each view, as well as the camera intrinsics. This can be repeated for each camera in the calibration setup.

Although Zhang's method is able to give us pose estimates by itself, these estimates are completely independent from view to view. It is unable to understand or estimate structural constraints on the geometry of the calibration setup such as the separation between the cameras of a rigid camera cluster, or the location of the center of a joint that a camera is mounted on.

Kinematic estimation. To solve this, we have a kinematic estimation stage which incorporates rigidity constraints using our SL-R algorithm. Given a description of the system as a kinematic tree, and the estimates from the camera calibration stage, our kinematic estimation determines an estimate for each label transformation automatically.

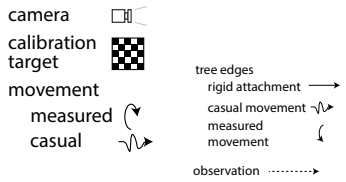
Nonlinear optimization. The result of our kinematic estimation may not be optimal in terms of reprojection error across all images. Our final stage takes the estimates from the kinematic estimation and the intrinsic estimates from the camera calibration. It performs nonlinear optimization over the relative poses in

the kinematic tree and any intrinsic parameters to produce a locally least-squares optimal solution.

2.4.2 Caliber problem definition

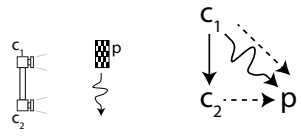
Although SL-R is a useful representation from an theoretical point of view, it has two shortcomings from a usability point of view. First, it can be cumbersome and unintuitive for a user to directly specify a complete instance of SL-R and interpret the solution. Second, SL-R does not define an objective function and parameterization on which to perform nonlinear optimization.

To address these, we observe that the inherent geometric structure of a broad category of camera systems can be described using a **kinematic tree**, akin to a scene graph, that expresses the relationships between objects (e.g. cameras, calibration targets, robot joints) in terms of nested frames of reference. See Figure 2.4 for examples. The user of Caliber describes the calibration problem by specifying this kinematic tree, indicating which transformations in the tree are known and which need to be solved for, and providing a set of camera observations that link nodes containing cameras to other nodes containing objects observed by the cameras. This description is equivalent in power to the SL-R representation, but is more intuitive for a user to specify, and defines a desired parameterization for the optimization stage. As we shall see in Section 2.4.4, this representation also provides an advantage in creating instances of SL-R that are solvable by our algorithm.



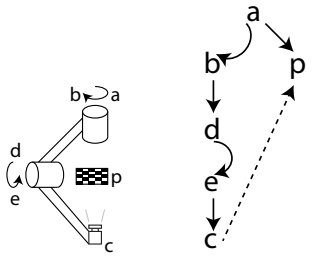
Setup Tree

(a) **Legend.**



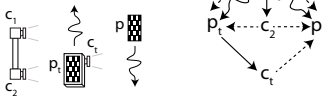
Setup Tree

(b) **Stereo** pair.



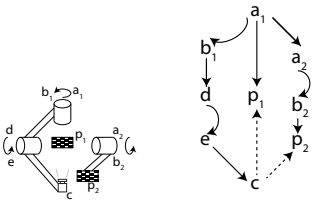
Setup Tree

(c) Spherical **gantry.** The camera arm has two revolute joints with different axes of rotation.



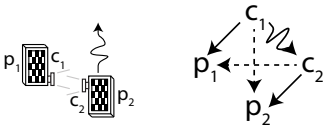
Setup Tree

(d) Stereo pair with **tablet.** The goal is to calibrate the camera and display on a tablet computer.



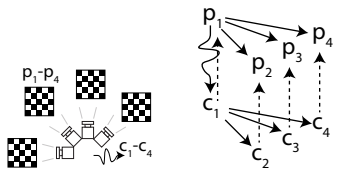
Setup Tree

(e) **Two-arm** spherical gantry. The second arm has only a single axis of rotation.



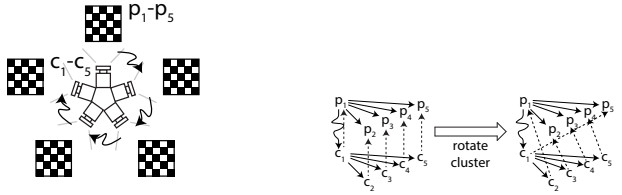
Setup Tree

(f) **Facing** smartphones. These are two devices with screens and front-facing cameras that take photographs of each other.



Setup Tree

(g) **Ad-hoc** cluster.



Setup Tree

(h) Point Grey **Ladybug.** The symmetrical arrangement of the cameras allows us to rotate the cluster so that each camera sees the next target in turn. Simulation suggests this improves the calibration. Here we have drawn the same tree to show the change in the source and target nodes of the observations after such a rotation.

Figure 2.4: Calibration setups and their corresponding trees. Drawings not to scale.

Components

The kinematic tree is defined by the following components:

Nodes. Nodes represent objects in the system—cameras, calibration targets, the two sides of a robot arm joint, and so forth. Each node has exactly one parent, except for a single root node which represents the global, "laboratory" frame. The nodes thus form a tree. Each node can also hold any other variables, such as intrinsic parameters, that are not used in the kinematic estimation stage but will be optimized over in the nonlinear optimization stage.

Observations. In addition to nodes, we have observations. An observation is defined by the following:

- A source node (representing a camera) and a target node (representing a calibration target).
- An estimate of the relative pose between the source and target node. This produces a relative pose constraint.
- The state of the system at the time the observation was taken. Specifically, this is defined by relative poses between each pair of adjacent nodes on the paths from the root to the source and target nodes. Each of these can be initially known, in which case they are specified as a rigid transformation value, or initially unknown, in which case they get a specified label as in the rigidity constraints of SL-R. Labels can be reused across different observations, resulting in shared rigidity constraints. For example, if one observation labels

a relative pose "A", and another observation labels also labels a relative pose "A", then those two relative poses are constrained to have the same value.

- The reprojection error function of that view, defined in terms of the relative pose between its source and target nodes and any variables such as intrinsic parameters. See 2.4.5 for details.

The kinematic estimation stage constructs and solves an instance of SL-R using the relative pose information provided in the observations. Then, the nonlinear optimization stage optimizes over the aggregate of the reprojection errors for all observations.

Note that although Caliber uses cameras, it is not bound to them: any measurement device that can produce a relative pose estimate would work. Likewise, reprojection error is just one option for an objective function; any objective function defined in terms of the relative pose between the source and target nodes and some intrinsic parameters could be used.

2.4.3 Camera calibration via Zhang's method

Caliber uses Zhang's algorithm, via Bouguet's [6] implementation, to get rough estimates of relative poses between cameras and targets and of the camera intrinsics. Bouguet takes a set of photographs of a checkerboard calibration target from a single camera, determines the 2D position of grid intersections in each, and uses these to estimate the relative pose between the camera and the target in each photograph, as well as a single set of camera intrinsics for the entire set. Therefore, we run Bouguet over all of the photographs that came from each camera to produce

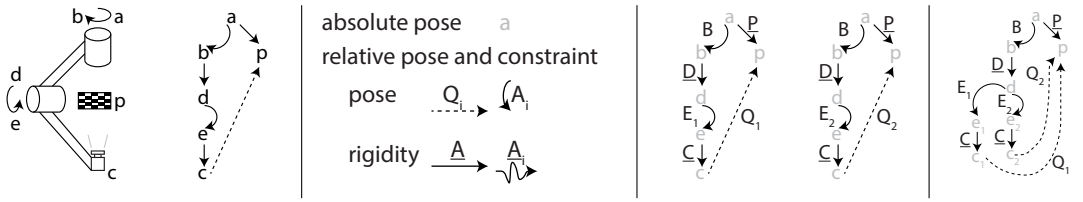


Figure 2.5: Reduction from the kinematic tree to SL-R for the spherical gantry case of Figure 2.4 (c). **Far left.** Reproduction of the setup diagram and kinematic tree. **Center left.** Legend. Each node in the kinematic tree may produce more than one corresponding absolute pose. After the reduction, all edges become relative poses. Measured relative poses get a relative pose constraint, and non-measured relative poses get a rigidity constraint (underlined). **Center right.** Two observations of the spherical gantry. Each consists of a path from the root to a source and to a target node, and a cross-edge between the source and target nodes. In this example, the top joint transformation B remains the same but the bottom joint transformation E changes. **Far right.** The corresponding SL-R instance. The top part of both cycles map onto the same absolute poses since they are the same, whereas the differing transformation for E produces two absolute poses for each node below it. Observe that the cycle consisting of E_1, E_2, Q_1, Q_2, and the Cs corresponds to an equation of the form $AX = XB$.

a relative pose estimate between the camera and calibration target for each view, and intrinsic parameter estimates for each camera.

Although Bouguet's implementation is a useful building block, CALIBER is not tied to this particular implementation; it is simply a convenient way of generating relative pose estimates.

2.4.4 Kinematic estimation

The goal of the kinematic estimation is to produce an estimate for each label that is consistent with all of the observations. Our kinematic estimation stage reduces the kinematic tree to SL-R and solves it using our SL-R algorithm.

We perform our reduction according to the following correspondence between

the kinematic tree and an instance of SL-R:

- Each appearance of a node in an observation in the kinematic tree maps to an absolute pose in SL-R—though as we will explain shortly, multiple appearances of the same node may map to the same absolute pose.
- Each known or estimated relative pose (dotted "observation" and curved "measured movement" arrows in Figure 2.4) mentioned in an observation, including the estimated transformation between the source and target node, becomes a relative pose constraint in SL-R between the corresponding absolute poses.
- Each rigid unknown relative pose (straight solid "rigid attachment" arrows in Figure 2.4) between two nodes becomes a rigidity constraint between the corresponding absolute poses.
- We need not include the squiggly "casual movement" arrows in Figure 2.4 as their relative pose is not held fixed between observations.

To keep the number of nodes to a minimum and give our SL-R algorithm information about which absolute poses are identical, we use the structure of the kinematic tree to guide the construction of our instance of SL-R. The core idea behind our strategy is to consider each pose in the frame of the root node, and make sure that symbolically equivalent poses are assigned the same absolute pose in our instance of SL-R. To do this, we keep a dictionary of which absolute poses we have seen and reuse them if they are involved in multiple observations. We then connect each absolute poses using the constraints specified in the observations involving that absolute pose. Pseudocode is shown in Algorithm 2, and a pictorial example is shown in Figure 2.5.

Algorithm 2: Reduction to SL-R

```
absolutePoses = new dictionary: sequence of constraints from root to
 current node → absolute poses
foreach observation do
    absolute source pose = processPath(observation source)
    absolute target pose = processPath(observation target)
    add relative pose constraint between the absolute source and target
     poses with the observation's pose estimate
Function processPath(stNode)
    foreach node and relative pose on path from root to stNode do
        current absolute pose = absolutePoses[sequence of constraints from
         root to current node]
           (create new absolute pose if not already existing)
        if relative pose is known then
            add relative pose constraint between current absolute pose and
             its parent
        else
            add rigidity constraint between current absolute pose and its
             parent
    return currentAbsolutePose
```

2.4.5 Nonlinear optimization

Once the kinematic estimation is done, we have a complete set of estimates for the rigidity constraints. From here, given a desired parameterization of individual transformations in the tree, we can automatically compute the reprojection error across all images and its Jacobian with respect to those parameters and the intrinsic parameters of the cameras. We can then apply a nonlinear optimization method such as Levenberg–Marquardt to refine the answer. This is similar to bundle adjustment, but respecting the specified kinematic tree and its implied rigidity constraints. Note that the transformations initially estimated by the kinematic estimation stage are *not* held fixed during optimization: the optimization is allowed to jointly refine these transformations (subject to rigidity constraints) as well any other parameters such as camera intrinsics. The solution that minimizes least-

squares reprojection error is statistically optimal, in the sense that it is a maximum likelihood estimate of the parameters under the standard model of Gaussian noise in image space.

Objective function

The global objective function of the nonlinear optimization stage is the sum-of-squares of the reprojection error across all observations (defined in Section 2.4.2).

For each observation i, the reprojection error is the difference between the image points predicted by the projection function f_i, and the actually observed image points $\mathbf{y}_i$:

$$\mathbf{z}_i = f_i\left(p_1, p_2, \ldots, \mathbf{x}_i\right) - \mathbf{y}_i \tag{2.8}$$

where $\mathbf{x}_i$ is the coordinates of the calibration target's feature points (in the target frame) expressed as a matrix of column vectors and $p_1, p_2, \ldots$ are the parameters into the projection function. The nonlinear optimization minimizes the sum of squares of these $\mathbf{z}_i$ across all observed points and all observations.

The projection function can be further decomposed into two parts. The intrinsic part computes the predicted image points given a set of intrinsic parameters such as focal length, principal point, and radial distortion, and the coordinates of the target points in the camera's frame. The coordinates of the target points in the camera's frame are equal to their coordinates in the target frame $\mathbf{x}_i$, transformed by the relative pose P_{st} between the camera and the target. The other part of the projection function computes this relative pose between camera and target P_{st}

as a function of extrinsic parameters such as rotations and translations of relative poses in the kinematic tree.

$$f_i = f_{i,\text{int.}}\left(p_{\text{int.},1}, p_{\text{int.},2}, \ldots, P_{st}\mathbf{x}_i\right) \tag{2.9}$$

$$P_{st} = f_{i,\text{ext.}}\left(p_{\text{ext.},1}, p_{\text{ext.},2}, \ldots\right) \tag{2.10}$$

We will now discuss the specific parameters that we use.

Parameterization

The intrinsics can be parameterized as per the standard projection function of e.g. [27], whose parameters are focal length, principal point, and radial distortion.

As for the extrinsics, we parameterize the label transformations using three rotation and three translational parameters. The translation parameters are simply the three Cartesian coordinates of the translation. We use the axis-angle representation for rotations and parameterize the rotations as a second rotation relative to the rotation produced by the kinematic estimation. Symbolically, given an initial rotation estimate R_{KE} and a rotation parameter vector r, the net rotation is

$$R_{\text{net}} = R_{\text{KE}} e^{[r]_\times} \tag{2.11}$$

where $[r]_\times$ is the cross-product matrix corresponding to r and $e^{[r]_\times}$ is the axis-angle rotation implied by r. Since we expect the kinematic estimation to produce a good estimate for the rotation, we expect this second rotation to stay close to the identity throughout the optimization process.

Note that the full 6-D parameterization for each label transformation may be redundant in some cases. Here we rely on the regularization of nonlinear optimizers

such as Levenberg-Marquardt to keep the solution near the initial guess along the ambiguous degrees of freedom.

Jacobian

To perform nonlinear optimization efficently, we need to compute the Jacobian of the reprojection error. The Jacobian of the intrinsic parameters is straightforward to analyze and will not be covered here. This leaves the Jacobian of the extrinsics. Given an observation with camera node s and calibration target node t, the relative pose between them P_{st} is $P_s^{-1}P_t$. In turn, P_s is the composition of the poses on the path from the root to s, and likewise for t:

$$P_s = \prod_{S_i \text{ on path from root to } s} S_i \tag{2.12}$$

$$P_t = \prod_{S_i \text{ on path from root to } t} S_i \tag{2.13}$$

For any extrinsic parameter p we can determine $\frac{dP_s}{dp}$ and $\frac{dP_t}{dp}$ via the product rule. Then

$$\frac{dP_{st}}{dp} = \frac{d}{dp}\left(P_s^{-1}P_t\right) \tag{2.14}$$

$$= \frac{dP_s^{-1}}{dp}P_t + P_s^{-1}\frac{dP_t}{dp} \tag{2.15}$$

$$= -P_s^{-1}\frac{dP_s}{dp}P_s^{-1}P_t + P_s^{-1}\frac{dP_t}{dp} \tag{2.16}$$

gives us the derivative of the pose between the camera and the target P_{st} with respect to the extrinsic parameter p.

2.5 Evaluation and results

2.5.1 Evaluation

In evaluating the results of experiments, we need to distinguish between evaluating Caliber itself and evaluating particular calibration setups.

Evaluating Caliber

We wish to evaluate Caliber based on whether the locally least-squares optimal solution it produces is the globally optimal solution. Unfortunately, we know of no feasible method of proving whether a particular locally optimal solution is globally optimal. And while our ultimate goal is to be able to calibrate real systems, we cannot simply take a real system and compare Caliber's result to ground truth either: there is a chicken-and-egg problem in that the calibration system we are evaluating is itself the best, or even only, method of trying to determine the ground truth in the first place. Many of our setups were constructed by hand without *a priori* measurements, and even in setups involving purpose-built equipment, such as the two-arm spherical gantry, we found that the calibration can outperform construction specifications.

Therefore, to evaluate the result of Caliber compared to the globally optimal solution, we turn to synthetic data. We took Caliber's solution for each of our real experiments in 2.5.2, and used it as ground truth for a synthetic counterpart. Specifically, we reprojected all of the image points using the solution, added Gaussian noise with standard deviation matched to the residual of the Zhang calibration stage, and then used this synthetic data as input of a new calibration problem. The

ground truth solution therefore has RMS reprojection error approximately equal to the standard deviation of the noise added. Then we reinitialized and ran CALIBER on this new, synthetic calibration problem. In order to give a realistic amount of deviation of the initial pose estimates from ground truth, the reinitialization used the original Bouguet pose estimates rather the optimized pose estimates from the original solution. We expect the globally optimal solution to be close to the ground truth, and thus for it to have a similar RMS reprojection error as well. If our kinematic estimation is good, we expect our locally optimal solution to be close to the globally optimal solution, and, if the kinematic estimation falls within the basin of convergence of the global optimum, the locally optimal solution will also be the globally optimal solution.

In each case, the RMS reprojection error of CALIBER's solution matched the magnitude of the added noise to within three percent. Therefore, here the local solution produced by CALIBER is either globally optimal or at least statistically resembles such.

Evaluating a calibration scheme

Even if CALIBER does find the globally optimal solution, this does not mean that all calibration setups produce equally good results. Now we must define what constitutes a "good" result.

Problems with evaluating parameter values directly. One obvious possible definition of "good" is how close the solved parameter values are to ground truth. However, such an approach has the same problem of determining ground truth as discussed before.

It is also unclear what is the appropriate baseline to compare such errors to. For example, even within a single experiment the magnitude of the translational errors is subject to choice of units of measurement. Comparison of errors in parameter values between different setups is even more problematic; the parameters in one setup may not have the same interactions, the same units, or even the same dimensionality as those of another.

Cross-validation. Instead, we define "practically relevant" in terms of the predictive power of the results of Caliber. This has several advantages. First, it uses the same metric that Caliber is optimizing for, namely reprojection error. Second, the ground truth of this definition is something that we directly measured, namely detected image points. Finally, reprojection error is always measured in pixels, allowing an "apples-to-apples" comparison between the results of different setups.

To evaluate the predictive power of Caliber, we used leave-one-out cross-validation (LOOCV) for both synthetic data (as above) and experimental data. This operated on each experiment as follows: for each view, we ran one trial of Caliber with that view omitted, predicted the image points for the omitted view, and recorded the errors in the predicted image coordinates. We then aggregated these errors across all trials to produce our **LOOCV prediction errors**.

In interpreting the results of this cross-validation, we need to distinguish between different components of this prediction error:

- **Random error.** This results from noise in the feature points of the view to be predicted. This component of the error would still exist even if the parameter values produced by Caliber were perfect.

- **Parameter error.** The feature points we used in each trial to calibrate the system are also noisy. This noise propagates into errors in the parameter values produced by CALIBER.
- **Systematic error.** Finally, our intrinsic and extrinsic models are imperfect: camera lenses may not fit our projection model exactly, robot arm links may flex, joints may have hysteresis and backlash, and so forth. Such weaknesses in our models produce systematic error.

We did not observe any obvious structure in the Zhang-only reprojection error; thus, we consider the systematic error in the Zhang result to be negligible. Since the Zhang model has a separate pose for each view, and is thus over-parameterized relative to the kinematic tree, we consider the parameter error in the Zhang result to be negligible as well. This leaves the random error in the feature points, which we estimate as the Zhang method's reprojection error over all images (i.e. not leaving one out). The synthetic data is generated as if there was no systematic error; therefore, we use the LOOCV prediction error for synthetic data as an estimate for the random error plus parameter error. Finally, we estimate the total error as the LOOCV prediction error for experimental data. The flow of data in our evaluation is summarized in Figure 2.7.

The machinery for both generating synthetic test data from an existing calibration and performing LOOCV is integrated into CALIBER itself, allowing it to be a tool for evaluating calibration schemes in addition to its primary use in performing the calibrations themselves.

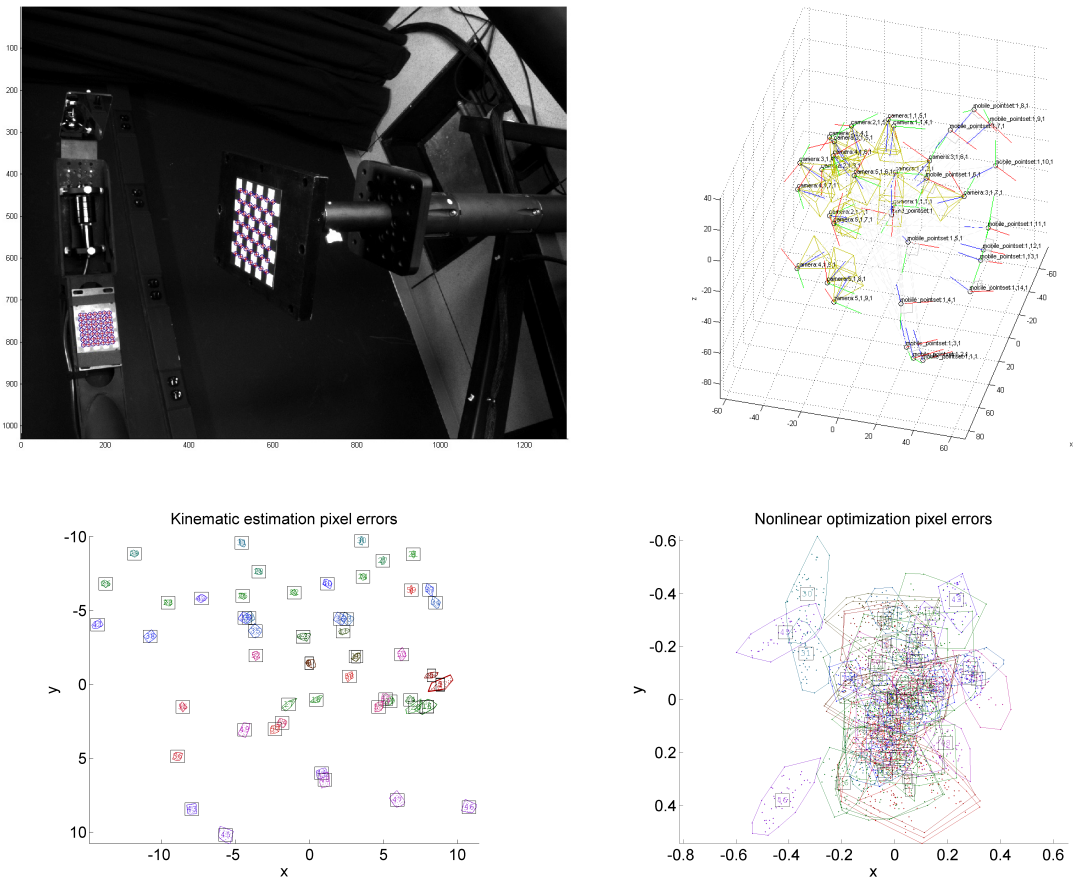


Figure 2.6: Example images from the calibration process for the two-arm case. See the supplemental material for more images of this and the other setups. **Top left:** Example input image. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. **Top right:** A depiction of the camera and calibration target frames for each observation. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets. **Bottom:** Pixel residuals after kinematic estimation (but before nonlinear optimization) and after nonlinear optimization. Colored pixels represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

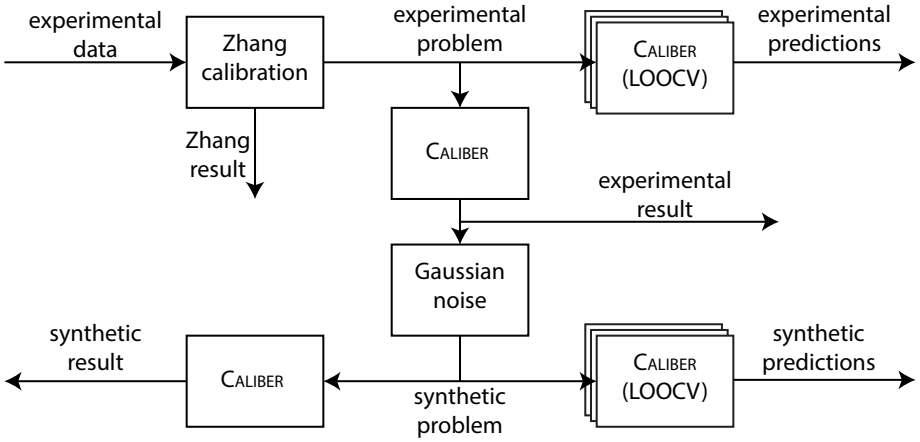


Figure 2.7: Flow of our evaluation scheme.

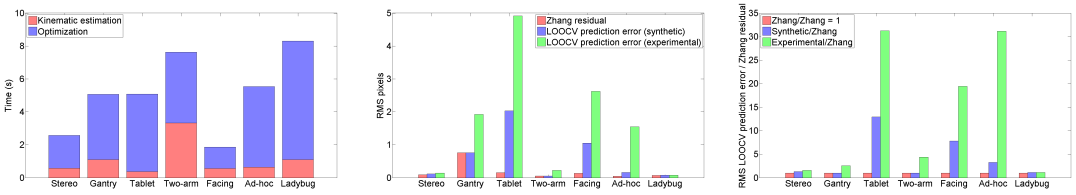


Figure 2.8: Results. **Left:** Kinematic estimation and optimization running time. **Center:** RMS residual for LOOCV prediction errors for synthetic and experimental data compared to the Zhang model by itself. Note that the Zhang model by itself uses independent poses for each observation and is thus strictly less constrained than our model. In particular, it does not obey pose or rigidity constraints and thus is non-predictive with respect to poses. Therefore, its residual is necessarily equal to or less than any model which *does* have to obey those constraints (such as ours). We include it here only as a baseline. **Right:** As center, but normalized by Zhang RMS residuals.

2.5.2 Experiments and results

Our experimental setups are depicted in Figure 2.4 and results are summarized in Figure 2.8. We ran each kinematic estimation and nonlinear optimization on an Intel i5-760 processor with 4 cores. Each calibration took less than ten seconds to compute. A brief discussion of each setup and its results follows. Example images for the two-arm setup from the calibration process are shown in Figure 2.6. Addi-

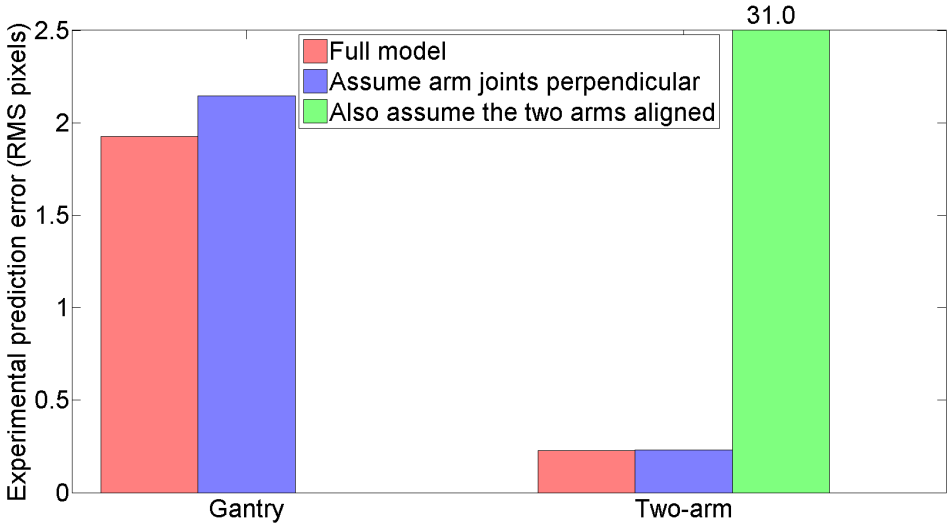


Figure 2.9: Comparison of different kinematic models for the gantry and two-arm cases. All of the models had the same prediction error for the synthetic data to within 3%. However, the prediction errors for the experimental data show that allowing CALIBER to solve for the relative pose between the two gantry arms produces a large improvement over assuming they are aligned, and allowing it to solve for the relative pose between the two camera arm joints produces a modest but noticeable improvement over assuming they are perpendicular.

tional details, results, and images for each setup can be found in the supplemental material.

Stereo pair

This is probably the simplest possible case that does not reduce to the basic Zhang method, and Bouguet has special-purpose code for this case, which provides a reference implementation to compare against. This made this a good sanity check for CALIBER.

We were able to duplicate the results of the Bouguet [6] calibration toolbox's special-case code for stereo cameras to within a margin that could be attributed

to machine precision, and the calibration performed well overall, with both experimental and synthetic prediction errors being only a modest factor larger than the Zhang residual.

Spherical gantry

This was the case that originally motivated this project. The gantry allows one to position a camera at any angle relative to an object to be photographed with high *precision*. However, calibration is required to make these angles highly *accurate* as well, which is important for applications such as reflectance measurement.

At first we assumed the two joints of the camera arm were indeed perfectly perpendicular. However, we found that allowing Caliber to solve for the relative pose between the two joints resulted in better predictions. The improvement was more obvious in the two-arm case, which used the same gantry. See Figure 2.9 for a comparison. This calibration showed an unusually high error in the Zhang residual (i.e. what appears to be random error), even compared to the two-arm case, but performed well overall. This may have been due to the camera we were using, which was different in the two cases.

Stereo pair with tablet computer

At its core, this setup attempted to measure the separation between a tablet computer's screen and camera. The tablet cannot see its own screen, so we viewed the tablet with a stereo pair, and added another calibration target that could be seen by all three cameras.

This was the most physically challenging experiment to perform due to the

difficulty of holding the tablet in place. The calibration was the worst of our cases overall in both experimental and synthetic data, with the views from the tablet computer being the worst within this setup, possibly because it could only be predicted using two other observations in serial.

Two-arm spherical gantry

For the reflectance measurement application, it is often useful to be able to change the lighting direction in addition to the viewing direction. This can be accomplished by mounting a light source on a second arm. Of course, this means that the second arm must also be calibrated to produce accurate angles.

As with the simpler gantry case, we at first assumed the two joints of the camera arm were indeed perfectly perpendicular, and furthermore that the axis of rotation of the second arm was aligned with the first. However, we again found that allowing Caliber to solve for the transformation between the two joints resulted in better predictions, as well as the transformation between the axes of the two arms. See Figure 2.9 again for a comparison.

The reprojection error for the solution does show some obvious structure, but in absolute terms the predictions for this setup were quite good for both synthetic and experimental data, suggesting that this setup is well-conditioned. More detailed plots of the reprojection error pattern as well as comparisons between the reprojection error for different models can be found in the supplemental material.

Facing smartphones

We wanted to see if we could calibrate two smartphones just by having them taking simultaneous pictures of each other. This type of calibration is appealing because it would not require any objects other than the devices to be calibrated. We used laptops instead of true smartphones since it was easier to set up a system for taking photographs without having to directly manipulate the devices.

Another purpose of this experiment was to demonstrate a practical, solvable case that could not be analyzed using our $AX = XB$ rule, but to our surprise Caliber determined that it actually was a case of $AX = XB$. However, the resulting calibration was one of the weaker ones in our set with relatively high prediction errors for both synthetic and experimental data, and more numerically stable setups might be sought.

Ad-hoc cluster

Calibrating clusters of non-overlapping cameras is useful for applications such as multiple cameras mounted rigidly on an autonomous vehicle. To model this, we bolted four cameras, including two DSLRs, onto a bar, mounted the cluster on a tripod, and placed a calibration target roughly in front of each camera.

This setup seemed to particularly suffer from systematic error. Given the weight of the DSLRs and only a single mounting point per camera, it seems likely this was due to the setup being less rigid than our kinematic model implied.

Point Grey Ladybug

We also tried a purpose-built cluster, the Point Grey Ladybug, which has featured in previous non-overlapping camera calibration work such as [40] (though that particular work used the Ladybug2 as opposed to our Ladybug3). We only calibrated the five side cameras, but the sixth/top camera could be included by mounting a calibration target in front of it.

The fields of view of the cameras do actually overlap slightly; however, in order to demonstrate that Caliber works in true non-overlapping cases, we did not take advantage of this fact. Indeed, including overlapping images actually *worsened* our cross-validation error. We believe this is because the photographs are highly distorted near the edges and corners, and the polynomial distortion model that we used does not handle this well, especially with a limited number of photographs. The edges and corners are exactly the parts of the photographs where overlapping views tend to occur.

The calibration of this setup performed much better than the facing and ad-hoc cases (the other non-overlapping or non-overlapping-like cases). We identify the following key advantages:

- We took a greater number of views with the Ladybug setup—more than twice as many as either of these other two cases.
- The Ladybug probably had better physical rigidity.
- The Ladybug setup had a greater number of cameras. Simulation (that is, constructing a synthetic scenario, adding noise, and running it through Caliber) suggests that increasing the number of cameras in a cluster im-

proves the calibration, even if the fields of view of the cameras remain non-overlapping and the number of views from each camera remains the same.

- The symmetrical arrangement of the cameras allowed us to rotate the unit so that each camera saw the next calibration target in turn. Again, simulation suggests that this improves the calibration even if no two cameras ever see the same calibration target at the same time and no camera ever sees more than one calibration target at the same time.

2.6 Conclusion

In this chapter we presented the sensor localization with rigidity problem (SL-R), which represents a class of localization problems involving not only relative pose constraints, but also rigidity constraints. Rigidity constraints encode the notion that the relative pose between two objects remains the same over time. We have shown that problem is NP-hard and gave an algorithm that is not guaranteed to produce a correct solution, but does so in practical cases, and runs in polynomial time.

Around our SL-R algorithm, we built Caliber, a practical calibration system for kinematically constrained camera systems that presents an easy-to-use kinematic tree representation to the user and refines the solution produced by our SL-R algorithm to a locally least-squares optimal result in terms of reprojection error. We demonstrated Caliber on a variety of both real and synthetic experimental data.

We will be releasing code to Caliber, and hope that it will be found useful both as a standalone tool and as a base for adaptations and expansions.

2.6.1 Future work

Possible extensions to this work include the following:

Reflections. Some existing methods, e.g. [40] use planar mirrors in their calibration systems. This could be a useful generalization of our system.

SL-R algorithm. We have found the efficiency and accuracy of our SL-R algorithm sufficient in practice. However, more efficient and accurate implementations may be possible. Our matrix multiplication-based implementation of the direct rule is the bottleneck for our current method, requiring a quadratic amount of storage and a n-by-n matrix multiplication for each of up to a linear number of outer loop iterations. This might be improved by collapsing connected components of known relative poses as the algorithm runs. We could also consider [26]'s Lie algebraic averaging in order to improve the accuracy.

AX = XB in the presence of noise. A more comprehensive analysis of $AX = XB$ in the presence of noise could lead to better accuracy and better decisions by our SL-R algorithm. For example, we might replace our discrete $AX = XB$ cases with a "soft" regularization that produces a finer-grained priority for which label to solve next.

Input recommendation. We could imagine an algorithm that recommends additional observations that may result in a better calibration. A sufficiently good such algorithm might even be used to automate the physical calibration process itself by directly controlling which observations are taken.

Input simplification. While our system frees the user from working out a complete calibration algorithm, it does require the user to input a description of the calibration setup's layout. Calibration could be made easier if this could be determined semi- or fully- automatically.

2.7 Appendix: Systems of AX = XB equations

In this section we cover solutions to systems of $AX = XB$ equations of rigid transformations. These solutions are not new in an of themselves, but they are a subroutine that allows our algorithm to solve non-trivial instances of SL-R.

2.7.1 Overview

The problem of solving a system of $AX = XB$ equations of rigid transformations is defined as follows: given a set of pairs of 4x4 rigid transformation matrices A_i, B_i, determine a 4x4 rigid transformation X such that for all i, $A_iX = XB_i$. Systems of equations of this sort naturally arise from situations such as hand-eye calibration and non-overlapping views from rigidly attached cameras.

2.7.2 Separation of rotation and translation

We opt to separate the $AX = XB$ equation on full rigid transformations into its translational and rotational parts. This gives

$$R_A R_X = R_X R_B \tag{2.17}$$

$$(R_A - I)\,\mathbf{t}_x = R_X \mathbf{t}_b - \mathbf{t}_a \tag{2.18}$$

2.7.3 Screw theory

It is also useful to observe that these equations describe a similarity transformation $B = X^{-1}AX$. Furthermore, [15] gives a method of characterizing $AX = XB$ systems using screw theory. Briefly, screws are a way of representing rigid transformations. They consist of a screw axis, which is a line in space, a translation magnitude along that axis, and a rotation magnitude about that axis. Screws have the property that the translation and rotation magnitudes are invariant with rigid transformations, whereas the screw axis transforms with rigid transformations. Therefore, if we have one or more $AX = XB$ equations, we can think of this problem as one where we attempt to find the transformation X that takes the screw axes given by $A_1, A_2, \ldots$ to those given by $B_1, B_2, \ldots$

2.7.4 Cases with rotation

Nonparallel (skew or intersecting) screw axes (0 undetermined dimensions). [61] gives a method to compute the unique answer in the case where the screw axes are nonparallel and the rotation magnitudes are not near the identity or a 180-degree rotation (so the screw axes are well-defined). If the problem is overconstrained by having more than two $AX = XB$ equations, they give a least-squares solution. In particular, they compute the rotational matrix that transforms

a $3\times n$ matrix of screw axis direction vectors α (corresponding to the As) to another one β (corresponding to the Bs) as

$$M = \beta\alpha^T \tag{2.19}$$

$$R = \left(M^T M\right)^{-\frac{1}{2}} M^T \tag{2.20}$$

After the rotation is solved for, they find a least-squares solution for the translation. We use this solution directly for the fully-determined case, and our solutions for the other cases are based on this method.

Parallel screw axes (1 undetermined dimension). [15] shows that in the case where the screw axes are parallel but not coincident, the rotation can be uniquely determined, but the translation has one undetermined dimension. For two such pairs of rigid transformations A_1, B_1 and A_2, B_2, we can construct three orthogonal directions using the (mutual) screw axis direction, the direction separating the two screw axes, and their cross product. We can then solve for the rotation as before.

Since R_A always has at least one eigenvalue of unity, $(R_A - I)$ in the translation equation 2.18 has a one-dimensional null space, and the solution space has one undetermined dimension. Namely, this is the common screw axis direction. Since all the screws are parallel, this null space is the same for all of the equations. In this case, we attempt to keep the translation as close to the origin as possible, which we do by adding another row to the matrix $\mathbf{r}_a^T \mathbf{t}_x = 0$. Finally, we stack the equations produced by each pair of $AX = XB$ equations to produce a single least-squares solution for $\mathbf{t}_x$.

Note that this gives an alternative method for solving the skew screw axis case: choose one screw axis as one direction, the direction along the shortest line between two screw axes as another, and their cross product as the third. Since in this case the null space of $(R_A - I)$ for different equations are not the same, the translation has a unique solution.

Coincident screw axes (2 undetermined dimensions). In the coincident screw axis case, only one direction for the As and for the Bs can be determined. The requirement that R_X rotate one direction onto another lets us solve for two rotational dimensions, but the third dimension is undetermined. In this case we choose the rotation axis to be along the (average) cross product of the screw axes of each A_i, B_i pair, which minimizes the rotation angle. We then solve for the translation as per the parallel axis case.

2.7.5 Cases without rotation

In some cases we may not have rotations at all; here the screw axis has direction equal to the translation but its offset perpendicular to the axis direction is undefined. As such, we may be able to construct axis directions, but the origin position of the axes are undetermined. In terms of the separated equations, the rotation equation 2.17 reduces to $R_X = R_X$, which gives us no information. The left side of the translation equation 2.18 is zero, which means that $\mathbf{t}_x$ cannot be determined at all, but there may be some information about R_X.

Nonparallel translation (3 undetermined dimensions). If the translations are nonparallel, we can use two non-parallel translations and their cross product to

construct three mutually orthogonal axis directions. This allows us to determine the rotation uniquely, as in the noncoincident screw axis cases. As noted above, the translation is completely undetermined.

Parallel translation case (4 undetermined dimensions). Similarly, the parallel translation case is analogous to the coincident screw axis case in that the rotation can be determined up to one dimension. As in the coincident screw axis case, the requirement that R_X rotate one direction onto another lets us solve for two rotational dimensions, but the third dimension is undetermined. We can use the same method as the coincident screw axis case to determine a feasible rotation. As in the nonparallel translation case, the translation is completely undetermined.

No information case (6 undetermined dimensions). If we have no information at all (no equations, or all As and Bs are the identity), then we simply guess X to be the identity.

2.7.6 Omitted cases

[15] addresses the cases where some or all of the rotations are 180 degrees. We have not implemented these cases so far because they are rare in practical use. Typically, the As and Bs in the equations result from the relationship between two positions of a joint; this means skew screw axes for a ball joint, coincident screw axes for a revolute or cylindrical joint, and parallel translation for a prismatic joint. Therefore, it is rare to find cases with only 180-degree rotations or mixed types of equations.

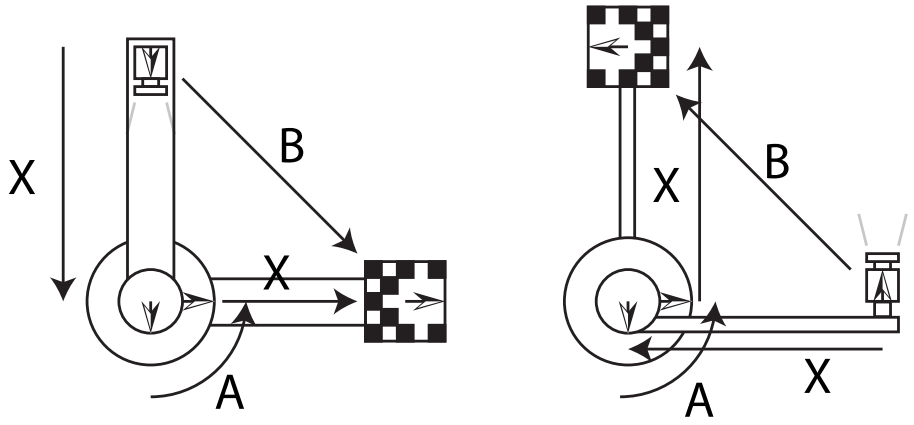


Figure 2.10: Two-fold ambiguity for $XAX = B$ illustrated using a camera, a calibration target, and a revolute joint. A is a pose measurement taken by a revolute joint, B is a pose measurement taken by a camera and a checkerboard, and the two Xs are known to be equal. There are two distinct possible values of X that are consistent with the measurements. On the left, X has no rotational component, while on the right, X has a 180-degree rotation.

2.8 Appendix: Complexity

In this section we show that SL-R is NP-hard. Our proof is of weak NP-hardness due to the assumption that a transformation matrix may be specified in constant space. Furthermore, we show that the uniqueness variant of SL-R—that is, given an instance of SL-R for which a solution is guaranteed to exist, does there exist more than one solution?— is also NP-hard. Therefore, if P $\neq$ NP, there exists no polynomial-time algorithm that can solve SL-R, nor determine whether an instance of SL-R is ambiguous.

Our proof has two parts. First, we construct a gadget that forces any solution to make a binary choice. Then, we use this gadget to construct a reduction from the NP-complete (unique) partition problem.

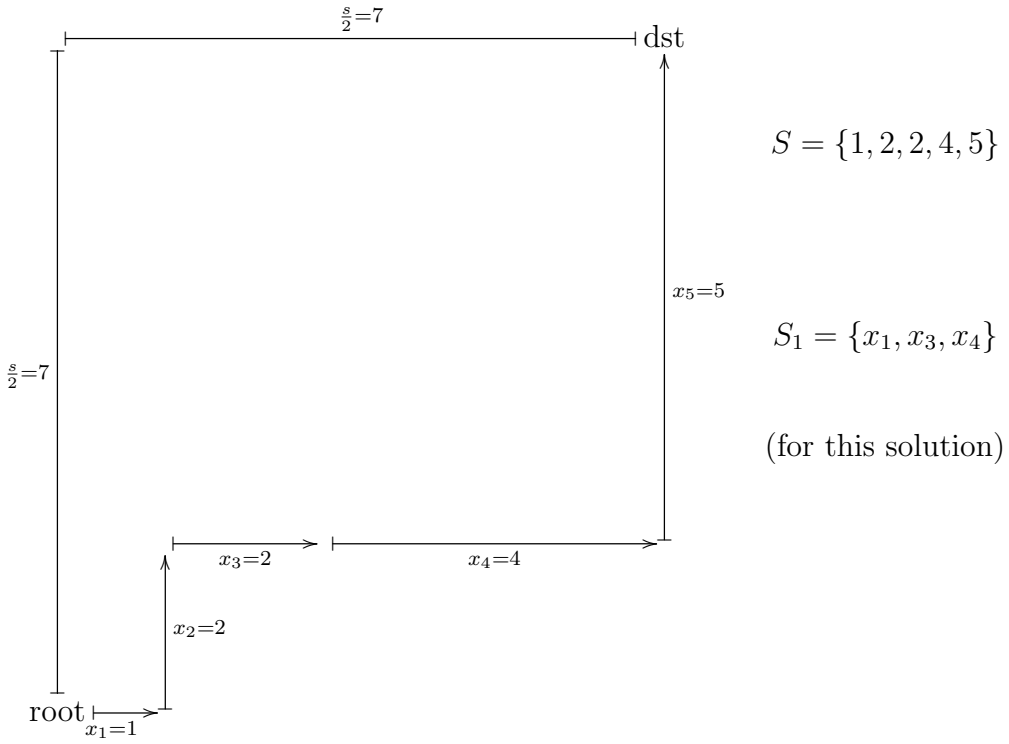


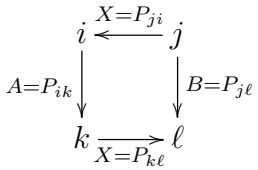
Figure 2.11: Pictorial example of a PP reduction to SL-R and its solution. We encode PP as a Manhattan walk where steps to the east represent one partition and steps to the north the other partition. We enforce the choice between these two directions using gadgets of the form $XAX = B$.

2.8.1 Binary choice via XAX = B

We begin by inventing a SL-R gadget that forces the solution to make a binary choice. This gadget relies on equations of the form $XAX = B$, where all three quantities are rigid transformations and X is to be solved for.

In terms of SL-R such an equation consists of four absolute poses i, j, k, ℓ with relative pose constraints P_{ik} and $P_{j\ell}$, and a rigidity constraint X on the relative poses P_{ji} and $P_{k\ell}$.

Pictorially, such an equation looks like this:



(2.21)

Mathematical properties. Inspiration taken from [29].

Suppose we represent each relative pose as a 4x4 rigid transformation matrix

$$\begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.22}$$

where R is a 3x3 rotation matrix, and $\mathbf{t}$ is a 3-vector representing translation.

From a single equation we have

$$XAX = B \tag{2.23}$$

$$AXAX = AB \tag{2.24}$$

$$AX = (AB)^{\frac{1}{2}} \tag{2.25}$$

Suppose we consider only the rotational part. For general real matrices, the square root has many possible values; however, since our solution is restricted to rotation matrices, this limits the possibilities. If $R_A R_B = I$, then I and any 180-degree rotation is a possible square root. Otherwise, $R_A R_B$ has a defined axis of rotation, and $(R_A R_B)^{\frac{1}{2}}$ must share that same axis, since applying any rotation twice preserves the axis of rotation. The magnitude of the rotation must be half that of $R_A R_B$, plus an integer multiple of π. Since a difference of 2π leaves the rotation matrix unchanged, this produces two possible values for $(R_A R_B)^{\frac{1}{2}}$ and therefore two possible values for R_X.

From here, we examine the translational part given the rotation. Denote the rotational and translational parts by R_X and $\mathbf{t}_x$ respectively. Then solving for the unknown $\mathbf{t}_x$ reduces to

$$R_X R_A \mathbf{t}_x + R_X \mathbf{t}_a + \mathbf{t}_x = \mathbf{t}_b \tag{2.26}$$

$$(R_X R_A + I)\, \mathbf{t}_x = \mathbf{t}_b - R_X \mathbf{t}_a \tag{2.27}$$

This has a unique solution as long as $R_X R_A + I$ is invertible. To see when this is the case, consider this product for a unit vector x:

$$x^{-1} (R_X R_A + I)\, x = x^{-1} R_X R_A x + 1 \tag{2.28}$$

The right side is only nonpositive when $R_X R_A$ is a 180-degree rotation. Apart from this case, $R_X R_A + I$ is positive-definite and therefore invertible. Thus, unless $R_X R_A$ is a 180-degree rotation, $\mathbf{t}_x$ has a unique solution given R_X.

The gadget. Suppose we have a cycle of rigidity constraints X and relative pose constraints A and B that produces an $XAX = B$ equation. Let the rotational parts of A and B equal

$$R_A = R_B = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.29}$$

From this we have

$$R_X = R_A^{-1} \left(R_A R_B\right)^{\frac{1}{2}} \in \{I, I'\} \tag{2.30}$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

$$I' = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.32}$$

Now let us examine the translational parts of A, B, and X. Suppose

$$\mathbf{t}_a = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.33}$$

$$\mathbf{t}_b = \begin{bmatrix} b \\ b \\ 0 \end{bmatrix} \tag{2.34}$$

for some constant b. In the case where $R_X = I$, we have

$$(R_X R_A + I)\,\mathbf{t}_x = \mathbf{t}_b - R_X \mathbf{t}_a \tag{2.35}$$

$$(R_A + I)\,\mathbf{t}_x = \mathbf{t}_b \tag{2.36}$$

$$\begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \mathbf{t}_x = \begin{bmatrix} b \\ b \\ 0 \end{bmatrix} \tag{2.37}$$

$$\mathbf{t}_x = \begin{bmatrix} b \\ 0 \\ 0 \end{bmatrix} \tag{2.38}$$

In the case where $R_X = I'$, we have

$$(R_X R_A + I)\,\mathbf{t}_x = \mathbf{t}_b \tag{2.39}$$

$$\begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \mathbf{t}_x = \begin{bmatrix} b \\ b \\ 0 \end{bmatrix} \tag{2.40}$$

$$\mathbf{t}_x = \begin{bmatrix} 0 \\ b \\ 0 \end{bmatrix} \tag{2.41}$$

Therefore, in our gadget, X either represents a step "forwards" ($+x$ direction) of size b, or a step to the "left" ($+y$ direction) of size b followed by an about-face. If $b = 0$, then the choice is between standing still and an about-face. A pictorial representation is shown in Figure 2.10. Also note that if b is an integer, all arithmetic here is integer, which shows that the NP-hardness of SL-R is not (solely) due to real arithmetic.

2.8.2 Reduction from the (unique) partition problem

With this gadget in hand, we now show a polynomial reduction from the (unique) partition problem to the (unique) sensor localization with rigidity problem.

The partition problem. Recall the partition problem (PP). The definition of the problem is as follows: given a set of positive integers $S = \{x_1, x_2, \ldots x_n\}$, determine if there exists a subset of the integers S_1 such that

$$\sum_{x \in S_1} x = \sum_{x \notin S_1} x \tag{2.42}$$

The standard form of the problem is well-known to be NP-complete; [19] shows that, given an instance of this problem for which it is guaranteed that at least one solution exists, the problem of determining whether there exists more than one solution (counting $S - S_1$ to be the same solution as S_1) is also NP-complete. They call this version the unique partition problem (UPP).

We likewise define the uniqueness version (USL-R) of SL-R to be: given an instance for SL-R which a solution is guaranteed to exist, does there exist more than one solution?

Integer representation. We represent the integers of PP in SL-R as follows. For each $x_i \in S$, we introduce two rigidity constraints X_i and X_i'. For both of these we construct an $XAX = B$ cycle of the form described above. For the one with X_i, we set $b = x_i$; for the one with X_i', we set $b = 0$. This produces four options for the product $X_i X_i'$, corresponding to our choice of picking $R_X = I$ or $R_X = I'$ for each rotational part:

1. A step forward of size x_i (choose $R_{X_i} = I$ and $R_{X'_i} = I$).
2. A step forward of size x_i followed by an about-face (choose $R_{X_i} = I$ and $R_{X'_i} = I'$).
3. A step left of size x_i (choose $R_{X_i} = I'$ and $R_{X'_i} = I'$).
4. A step left of size x_i followed by an about-face (choose $R_{X_i} = I'$ and $R_{X'_i} = I$).

We construct each such cycle so that they share a common "root" absolute pose, so these are the only choices to be made here. Since we do constant work and produce a constant number of poses, relative pose constraints, and rigidity constraints per element of S, this part of the reduction is linear in time and problem size.

A Manhattan walk. With our integer representation in place, we introduce one more cycle, which we will treat as two paths from the root to a destination. On one side we constrain the absolute pose of the the destination to equal

$$P_{\text{dst}} = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{2}s \\ 0 & 1 & 0 & \frac{1}{2}s \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.43}$$

$$\text{where } s = \sum_{i}^{n} x_i \tag{2.44}$$

On the other side we have a sequence of $2n$ poses with relative poses labeled such that they form the product

$$P_{\text{dst}} = \prod_{i}^{n} X_i X'_i = X_1 X'_1 X_2 X'_2 \dots X_n X'_n \tag{2.45}$$

This part of the problem reduction is also linear in time and problem size.

The intuition is that we have constructed a problem where the goal is to reach a destination $\frac{s}{2}$ north and $\frac{s}{2}$ east of the starting point, using a set of steps of size $x_1, x_2, \ldots x_n$.

At each two-pose step towards the destination $X_i X_i'$ a solution must take one of the four options enumerated earlier. This constrains the translations to lie only on cardinal directions. Since the Manhattan distance to our target is s and there is only have a total of s distance in translations in the product of rigid transformations, any solution must step towards the destination in every step. The solution cannot do an about-face before the last step, since it would have to step away from the destination on the next step, and the solution cannot do an about-face on the last step, since it would be facing the wrong direction at the end. Therefore any solution must consist of only options 1 and 3, with forward steps being to the "east" and left steps to the "north". Since the east steps total the same distance as the north steps in any solution, this forms a one-to-one correspondence to a solution to PP—the indices of the steps in which the solution moved east correspond to the indices of the integers in one half of the partition. To avoid the duplicate solution where S_1 is swapped with $S - S_1$, we require the first step to be east by adding a relative pose constraint.

An example is shown in Figure 2.11.

Therefore PP reduces to SL-R, UPP reduces to USL-R, and both SL-R and USL-R are NP-hard.

2.9 Appendix: Experiment details

A description of each setup follows. See Table 2.2 for more quantitative details. their corresponding kinematic trees.

2.9.1 Stereo pair

We mounted a pair of Point Grey Chameleon cameras on a rigid beam and took pairs of photographs of a checkerboard displayed on a LCD monitor at a distance of about 3 m from the cameras. We kept the cameras static throughout the experiment, while changing the position and orientation of the monitor between each pair of photographs. We took 12 pairs of photographs for a total of 24 views.

2.9.2 Spherical gantry

We mounted a Canon EOS 50D camera on a spherical gantry arm, which is a robot arm with two (nominally) perpendicular revolute (single-axis rotation) joints. The checkerboard was a printed pattern near the center of the gantry, about 0.5 m from the camera. We took 24 photographs.

2.9.3 Stereo pair with tablet computer

This experiment was the same as the stereo pair, except for the addition of an Apple iPad displaying a second checkerboard and having a third camera. The tablet was placed about 3 m from the stereo pair, while the monitor was placed about 2 m

farther, roughly in a line. Again we kept the stereo cameras static throughout the experiment, while changing the position and orientation of the monitor and tablet. We took 8 triplets of photographs, each corresponding to a single monitor and tablet configuration, with both the tablet and monitor checkerboards being visible (though sometimes partially obstructed by the tablet) in each of the observations from the stereo camera. Each set thus netted five views, for a total of 40 views.

2.9.4 Two-arm spherical gantry

This experiment used the same spherical gantry, but used a QImaging Retiga 1300i camera instead. It also involved a second checkerboard mounted on a second gantry arm. The second arm had only a single revolute joint, nominally aligned with the camera arm. The fixed checkerboard remained at about a distance of 0.5 m from the camera, while the arm-mounted checkerboard was at a distance of 1.1 to 1.3 m depending on the gantry configuration. We took 43 photographs; the fixed checkerboard was visible in 20 of them, and the arm-mounted checkerboard was visible in 39, for a total of 59 views.

2.9.5 Facing smartphones

We had two laptops with integrated cameras each display a checkerboard on their screen. We took 13 pairs of photographs from each laptop of the other laptop's displayed checkerboard, moving the laptops between each pair of photographs, for a total of 26 views.

2.9.6 Ad-hoc cluster

We bolted four cameras onto a rigid beam, facing outwards in a "fan" of about 90 degrees. Two of the cameras were the same Point Grey Chameleons from the stereo and tablet setups, but with different lenses, while the other two were DSLRs. Each camera observed its own checkerboards, displayed on a LCD monitor from 1.0 to 1.5 m away; no camera had any observations of any other camera's checkerboards. We took 6 sets of photographs from the camera cluster, moving the camera cluster between each set while keeping the monitors in place, for a total of 24 views.

2.9.7 Point Grey Ladybug

We used the five side-mounted cameras of a Point Grey Ladybug3 system. We mounted one checkerboard roughly in front of each camera. We moved the Ladybug between each set of exposures while keeping the checkerboards fixed. We took 13 sets of photographs from the five side cameras, for a total of 65 views.

Table 2.2: Description of experimental setups. Focal lengths are approximate. Numbers of views refer to the total number of views, not the number per camera. Checkerboard sizes refer to the number of feature points.

Experiment	No. views	Obs.			Checkerboards		
		Location	Resolution	Focal len.	Location	Size	Sq. size
Stereo pair	24	Left	1280x960 px	7000 px	Monitor	8x8	23.52 mm
		Right	1280x960 px	3500 px			
Spherical gantry	24	Gantry	4770x3177 px	13500 px	Fixed	8x6	9 mm
Stereo pair	40	Left	1280x960 px	7000 px	Monitor	8x8	23.52 mm
with tablet		Right	1280x960 px	3500 px	Tablet	7x10	19.05 mm
		Tablet	960x720 px	1200 px			
Two-arm,	59	Gantry	1300x1030 px	1380 px	Fixed	7x7	10 mm
spherical gantry					Arm	7x7	10 mm
Facing smartphones	26	Laptop 1	640x480 px	630 px	Laptop 1	8x8	13.80 mm
		Laptop 2	640x480 px	680 px	Laptop 2	8x8	15.00 mm
Ad-hoc cluster	24	Camera 1	4172x3168 px	4400 px	Target 1	8x8	23.52 mm
		Camera 2	1280x960 px	1650 px	Target 2	8x8	20.64 mm
		Camera 3	1280x960 px	2480 px	Target 3	8x8	15.00 mm
		Camera 4	4272x2848 px	3640 px	Target 4	8x8	21.12 mm
Ladybug	65	Camera 1-5	808x616 px	375 px	Target 1-5	6x8	35 mm

Table 2.3: Leave-one-out cross-validation results. Optimization residuals refer to the residuals when optimization is performed with all observations. LOOCV errors refer to the errors over all predictions in the leave-one-out cross-validation, where each observation is left out in one trial. If more than one calibration target appears in a single photograph, each calibration target in the photograph is considered as a separate observation. Running times refer to the optimization where all images were used. KE is short for kinematic estimation. The median and maximum columns refer to the absolute values of the residuals/errors. σ for the synthetic experiements refers to the residual of the Zhang calibration (independent pose estimates for every observation) and consequently the magnitude of the Gaussian noise of the synthetic experiments.

	Running time		Optimization Residuals (px)				LOOCV Errors (px)			
Experiment	KE	Opt.	RMS	Mean	Med.	Max	RMS	Mean	Med.	Max
Stereo pair	0.61 s	2.3 s	0.095	0.070	0.050	0.425	0.141	0.105	0.079	0.536
Spherical gantry	1.1 s	4.0 s	1.728	1.383	1.138	5.765	1.927	1.542	1.280	6.287
Stereo with tablet	0.39 s	5.1 s	0.381	0.199	0.101	3.960	4.911	2.038	0.414	25.817
Gantry, two-arm	3.3 s	4.3 s	0.186	0.146	0.119	0.617	0.229	0.177	0.139	0.786
Facing smartphones	0.79 s	1.4 s	0.144	0.110	0.088	0.631	2.622	1.984	1.457	8.064
Ad-hoc cluster	1.5 s	5.6 s	0.077	0.060	0.049	0.321	1.545	1.103	0.812	6.288
Ladybug	0.77 s	6.9 s	0.068	0.053	0.042	0.285	0.081	0.064	0.052	0.324

		Optimization Residuals (px)				LOOCV Errors (px)			
Synthetic experiment	σ (px)	RMS	Mean	Med.	Max	RMS	Mean	Med.	Max
Stereo pair	0.089	0.086	0.069	0.058	0.363	0.115	0.089	0.072	0.473
Spherical gantry	0.759	0.725	0.579	0.490	2.523	0.732	0.585	0.497	2.545
Stereo with tablet	0.157	0.156	0.125	0.105	0.600	0.969	0.328	0.148	4.885
Gantry, two-arm	0.052	0.051	0.041	0.034	0.199	0.052	0.041	0.034	0.199
Facing smartphones	0.135	0.136	0.108	0.092	0.485	1.075	0.816	0.673	3.893
Ad-hoc cluster	0.050	0.048	0.039	0.033	0.177	0.135	0.097	0.068	0.603
Ladybug	0.071	0.069	0.055	0.046	0.259	0.080	0.063	0.052	0.378

2.10 Appendix: Example images

This section gives three sets of images for each calibration setup:

1. An example photograph for each camera with the detected and reprojected feature points superimposed on them.
2. A plot of the camera and calibration target frames for every observation.
3. A plot of the reprojection errors before and after the nonlinear optimization.

2.10.1 Stereo pair

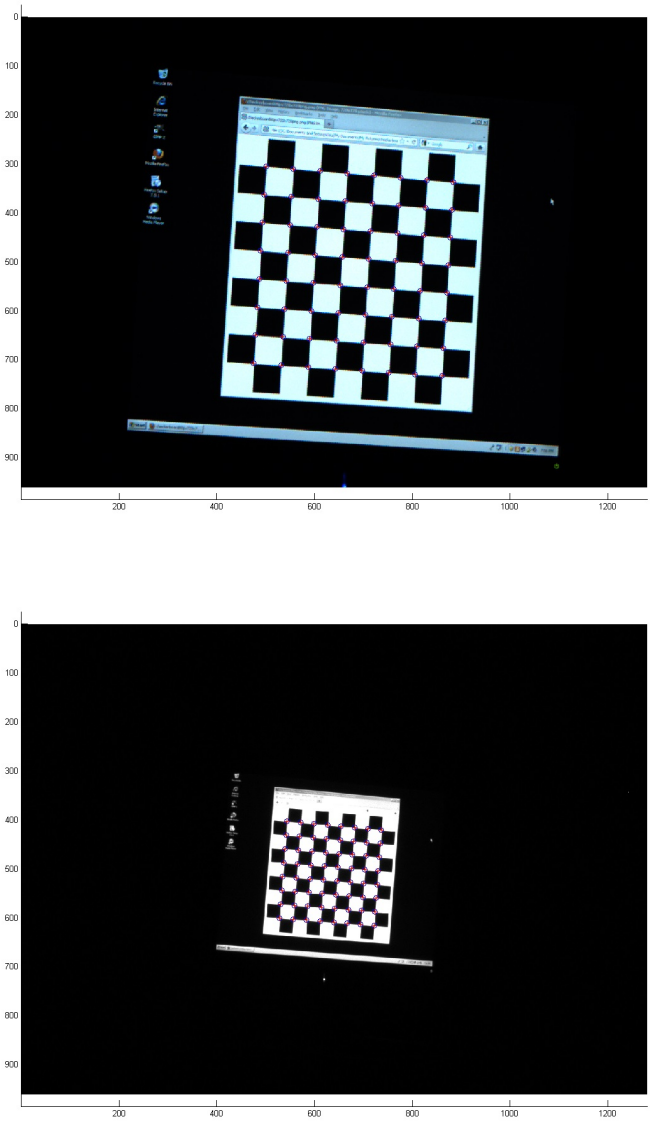


Figure 2.12: Examples of reprojections for the stereo pair experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. Both images are from the same configuration. The first image is from the left camera, and the second from the right camera.

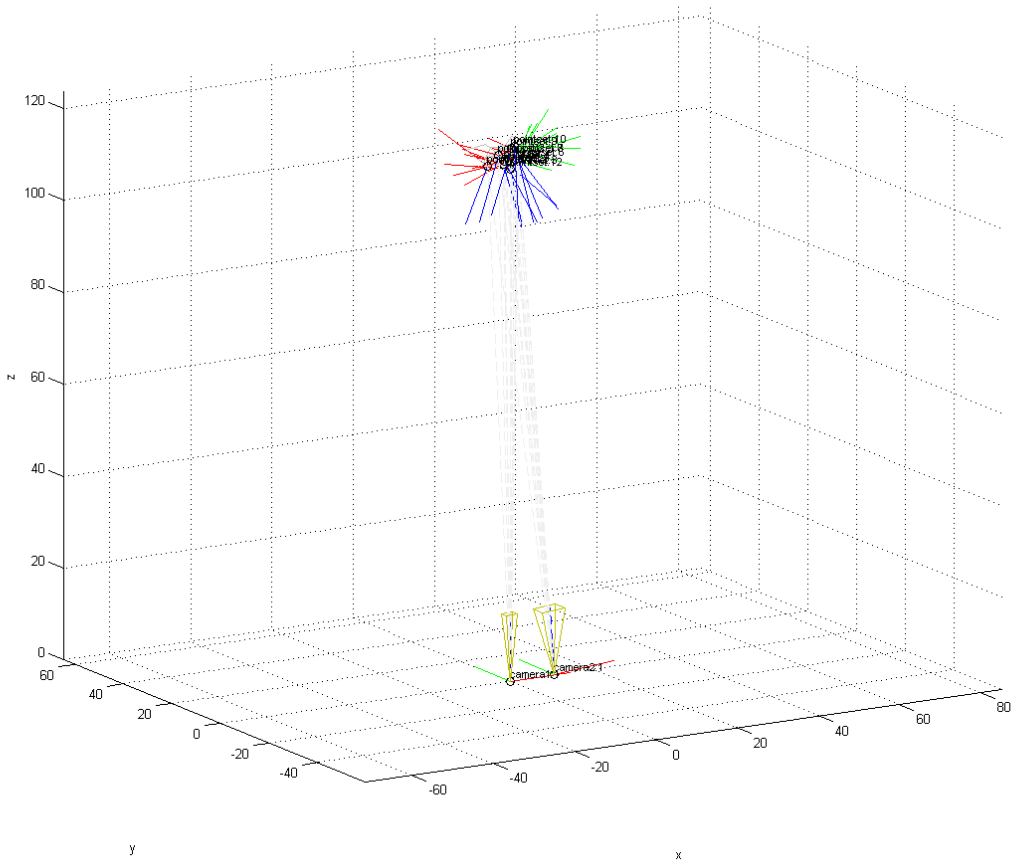


Figure 2.13: A depiction of the camera and calibration target frames for each observation for the stereo pair experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

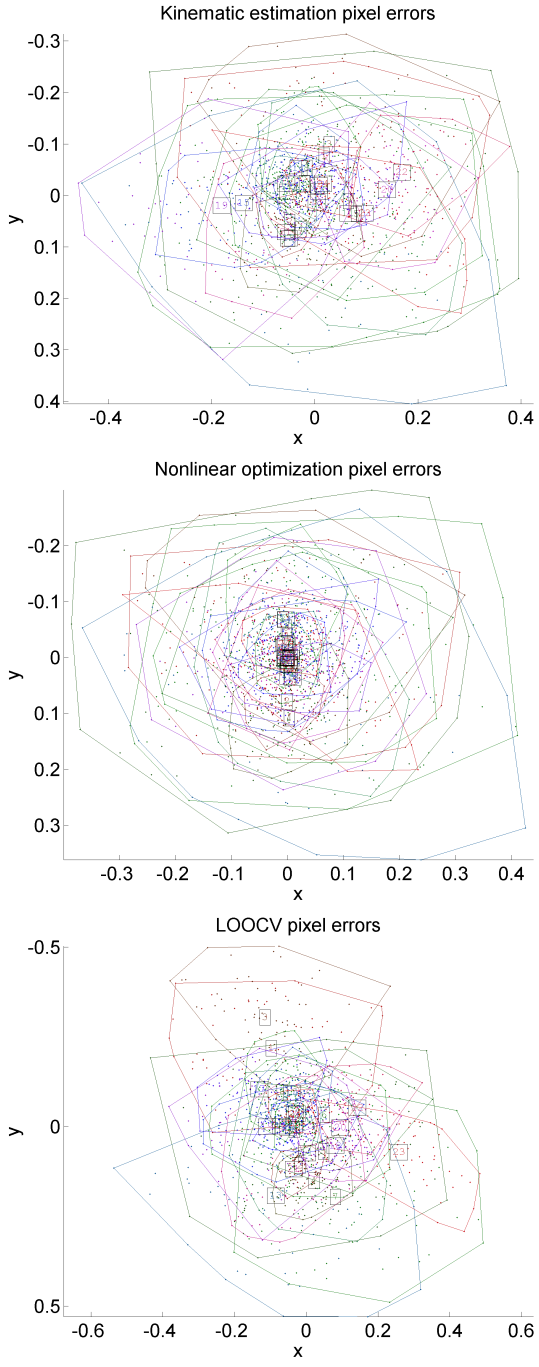


Figure 2.14: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the stereo pair experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

2.10.2 Spherical gantry

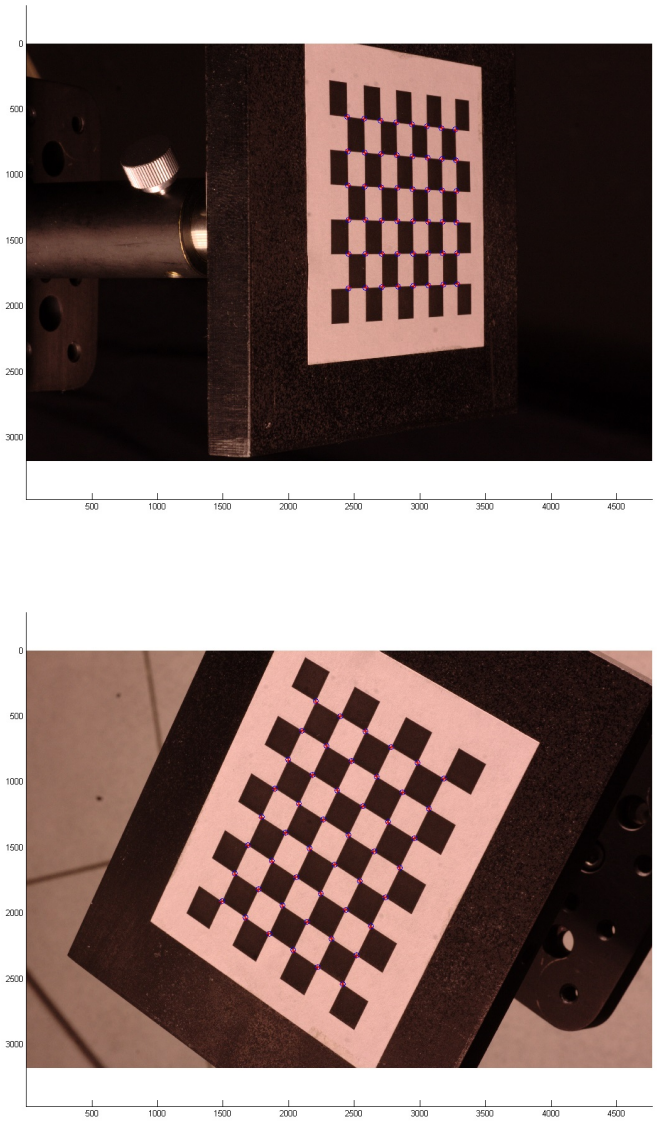


Figure 2.15: Examples of reprojections for the spherical gantry experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. The two images are from different positions of the camera arm.

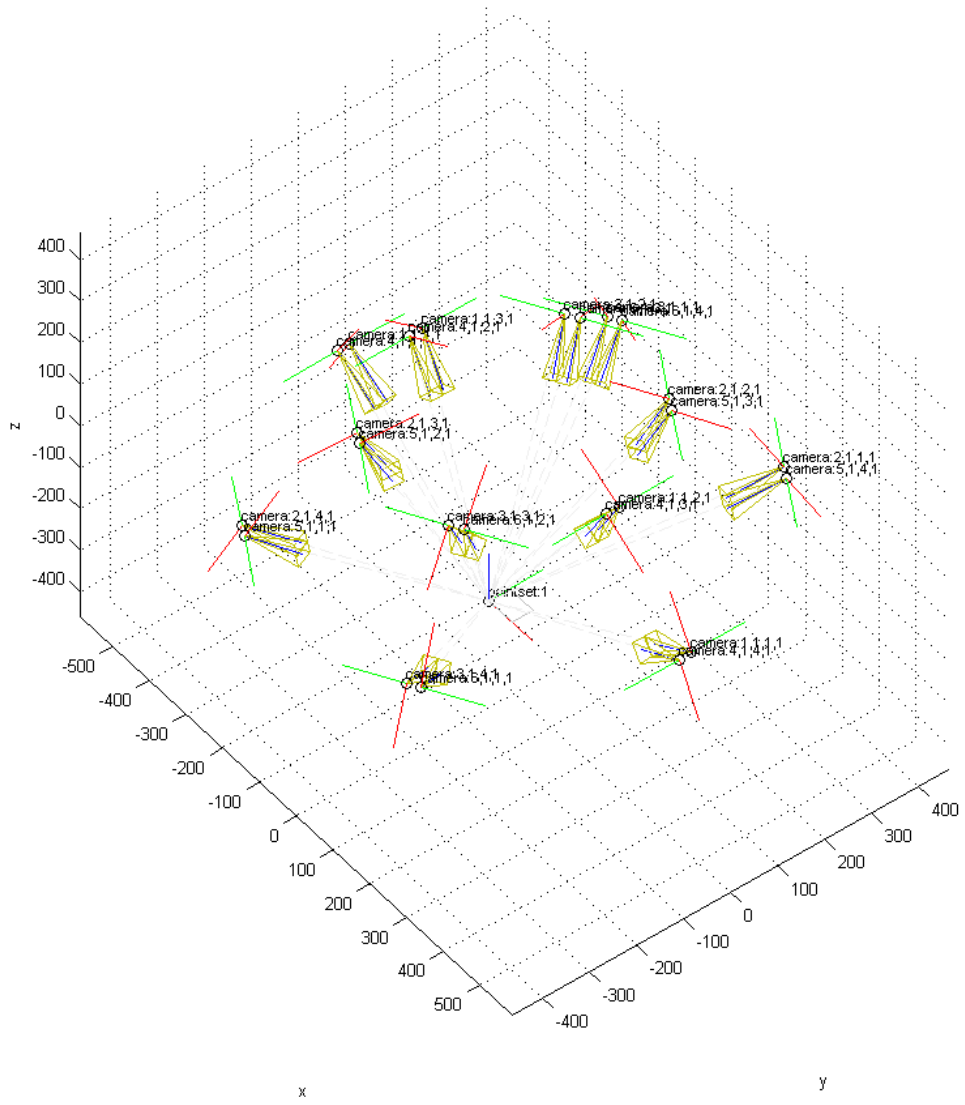


Figure 2.16: A depiction of the camera and calibration target frames for each observation for the spherical gantry experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

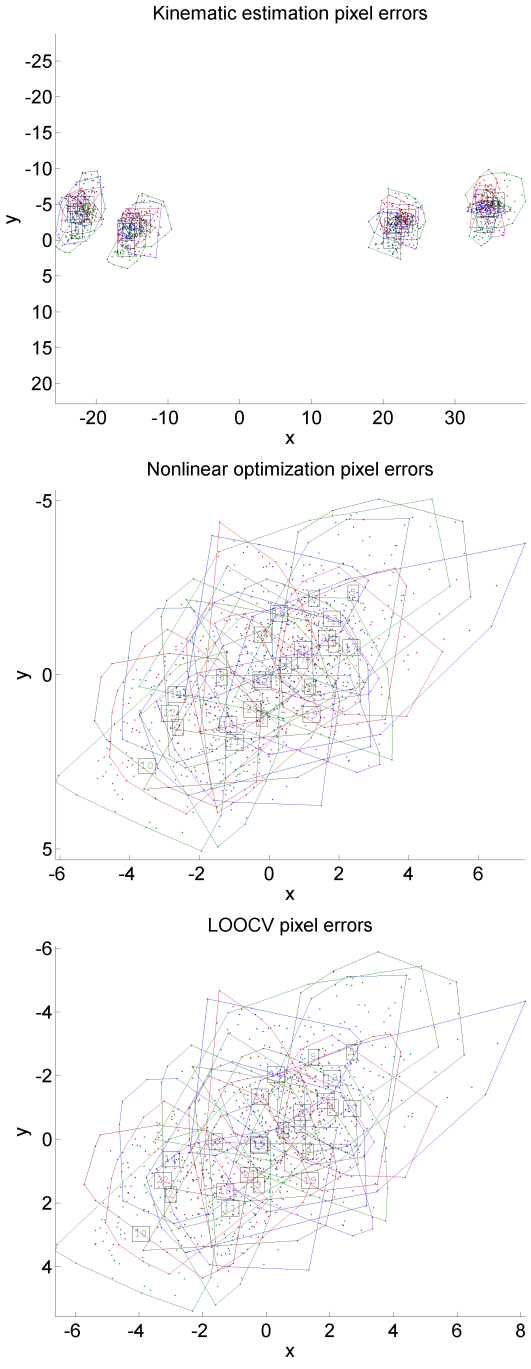


Figure 2.17: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the spherical gantry experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

2.10.3 Stereo pair with tablet

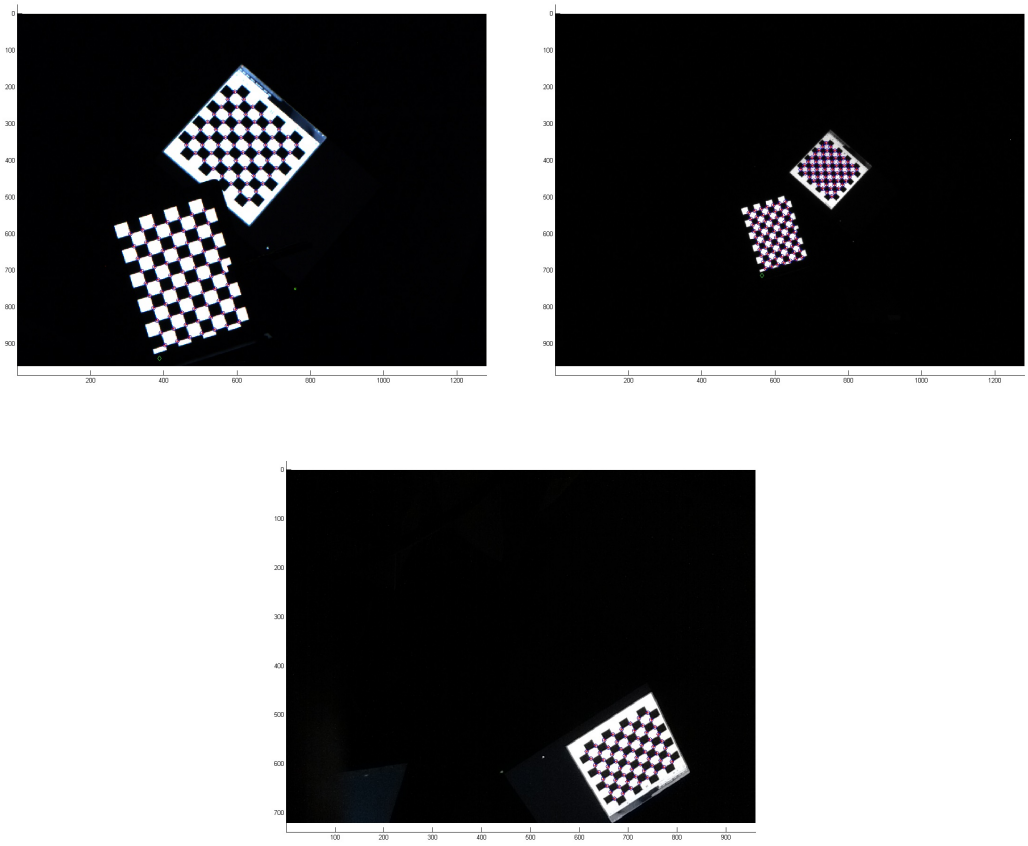


Figure 2.18: Examples of reprojections for the stereo pair with tablet experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. All three images are from the same configuration. The first image is from the left camera, the second from the right, and the third from the tablet. The reprojected tablet camera position in the first two images is indicated by a green diamond.

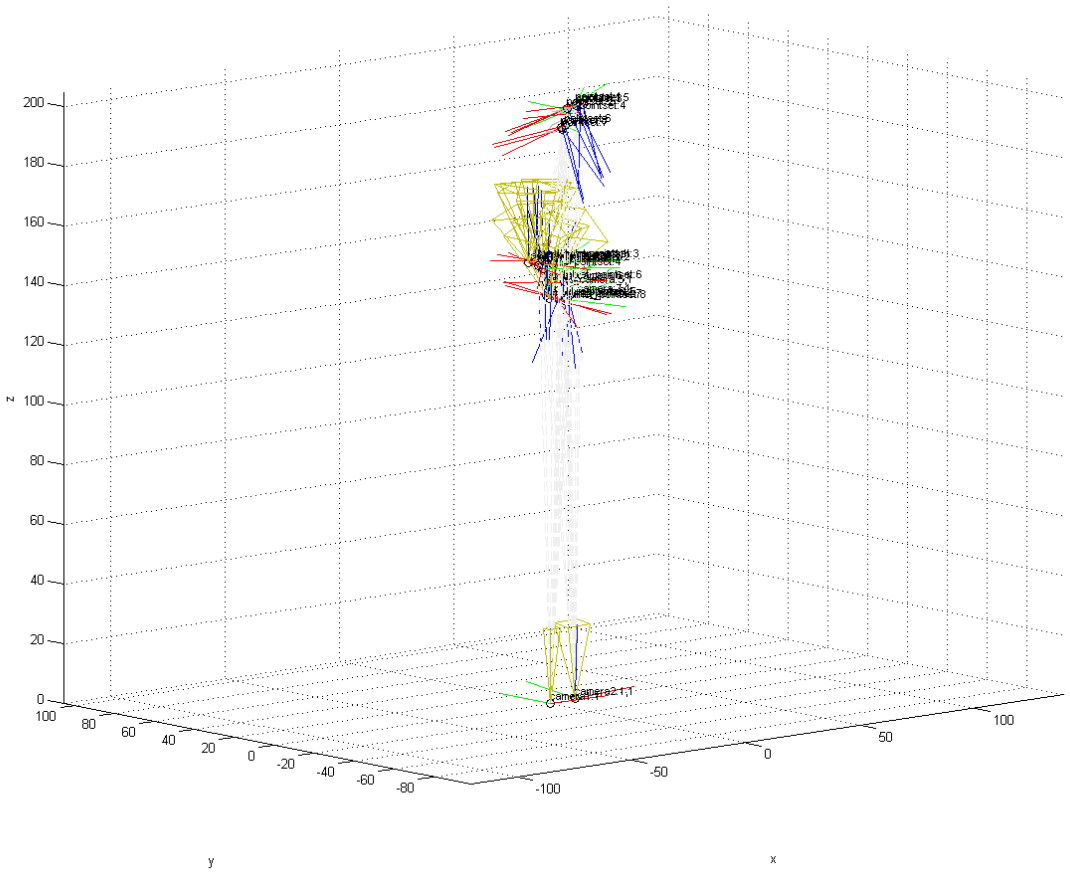


Figure 2.19: A depiction of the camera and calibration target frames for each observation for the stereo pair with tablet experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

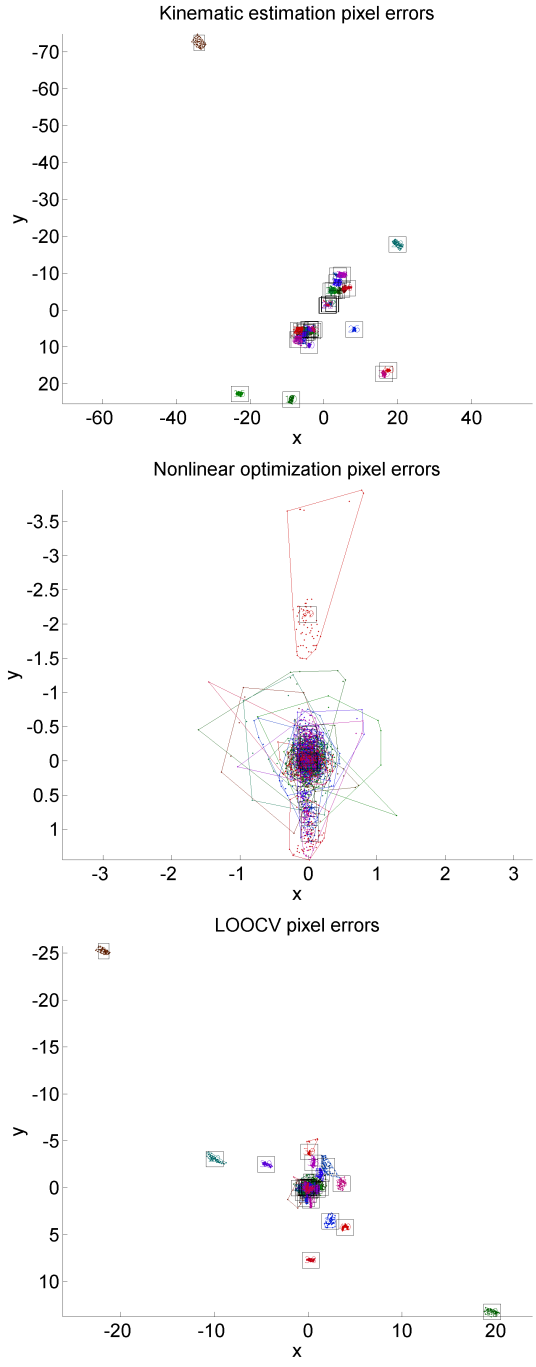


Figure 2.20: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the stereo pair with tablet experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

2.10.4 Two-arm spherical gantry

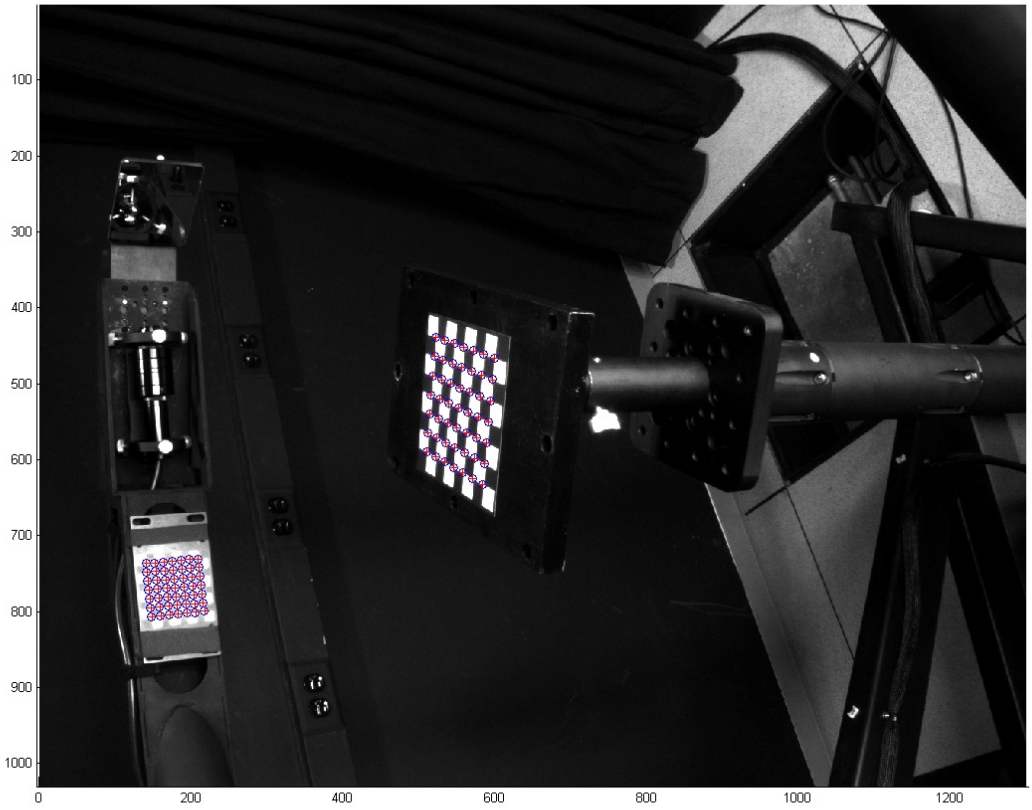


Figure 2.21: Examples of reprojections for the two-arm spherical gantry experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. Both checkerboards are visible in this image.

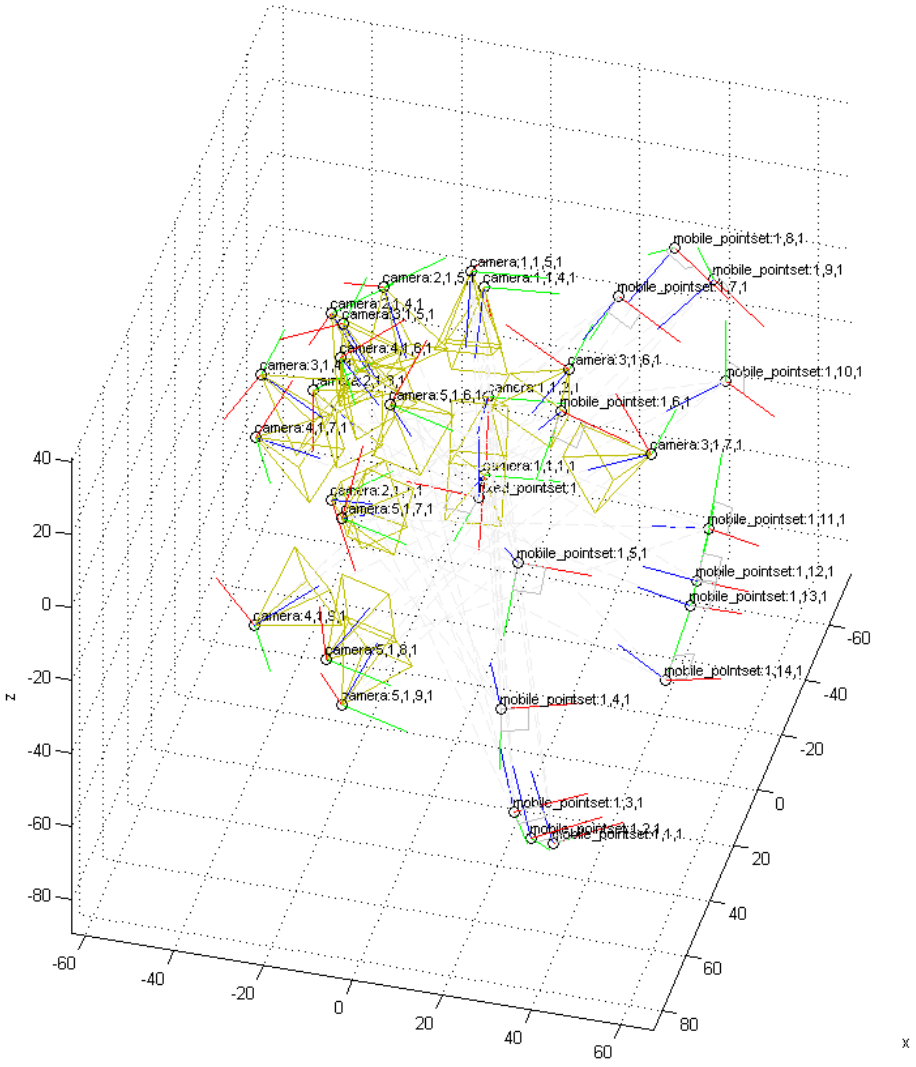


Figure 2.22: A depiction of the camera and calibration target frames for each observation for the two-arm spherical gantry experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

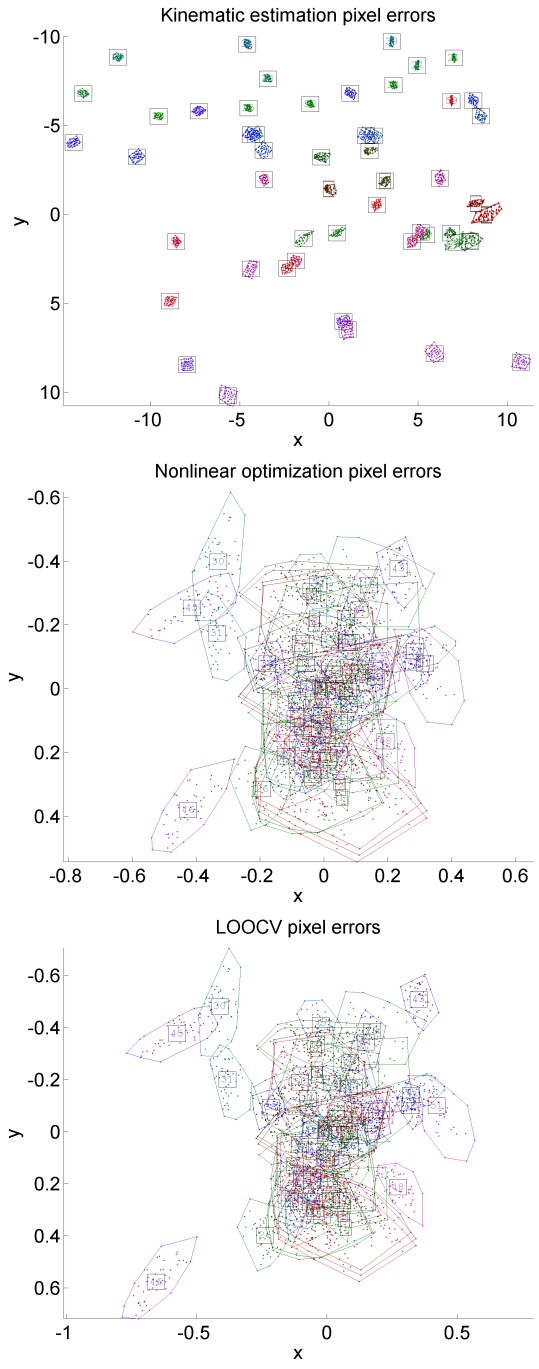


Figure 2.23: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the two-arm spherical gantry experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

Model variations

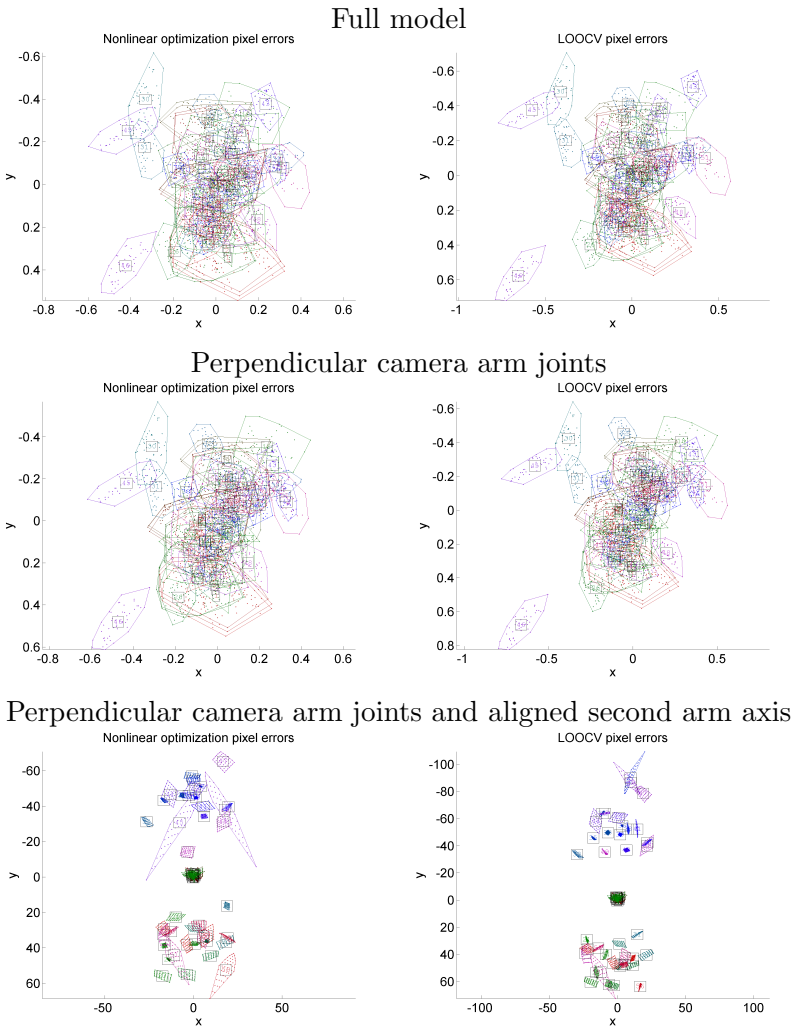


Figure 2.24: Pixel residuals for optimizer and LOOCV for three variations on the two-arm gantry model. The first row is the model we used. The middle row is for a model that assumes the two revolute joints of the camera arm are perpendicular. The last row additionally assumes the second arm's axis is aligned with the camera arm. This shows that our system can evaluate the strength of competing models: the joints of the camera arm are very nearly perpendicular, but dropping the assumption does give a very small improvement to the RMS LOOCV pixel residual. Meanwhile, the axes of the two arms are clearly not quite aligned.

2.10.5 Facing smartphones

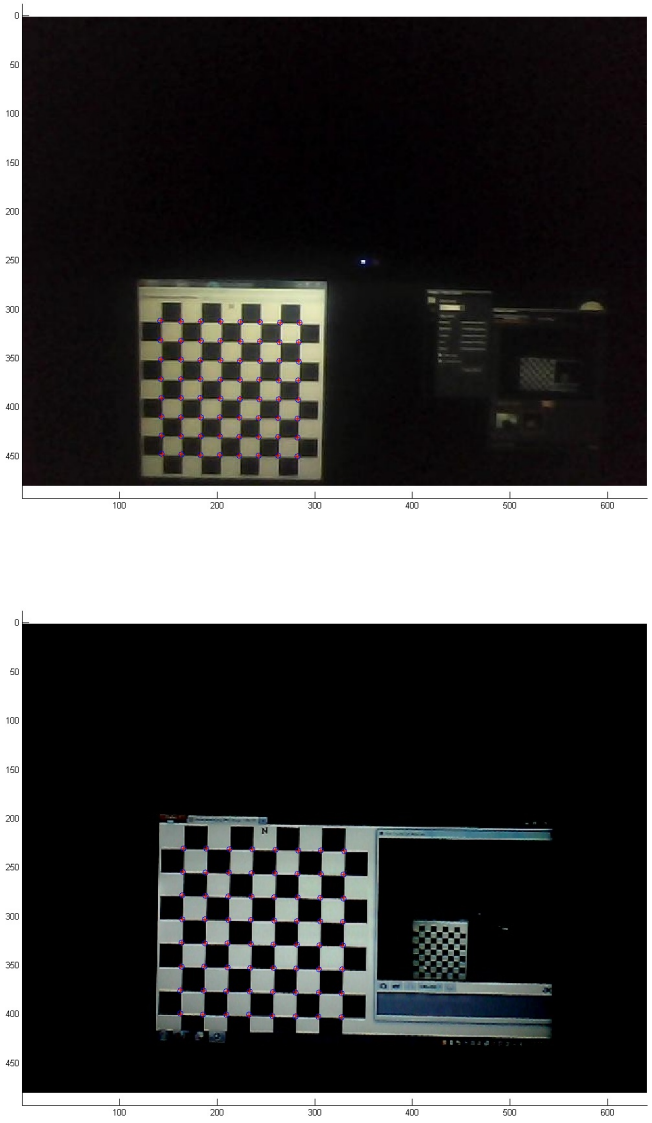


Figure 2.25: Examples of reprojections for the facing smartphones experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization.

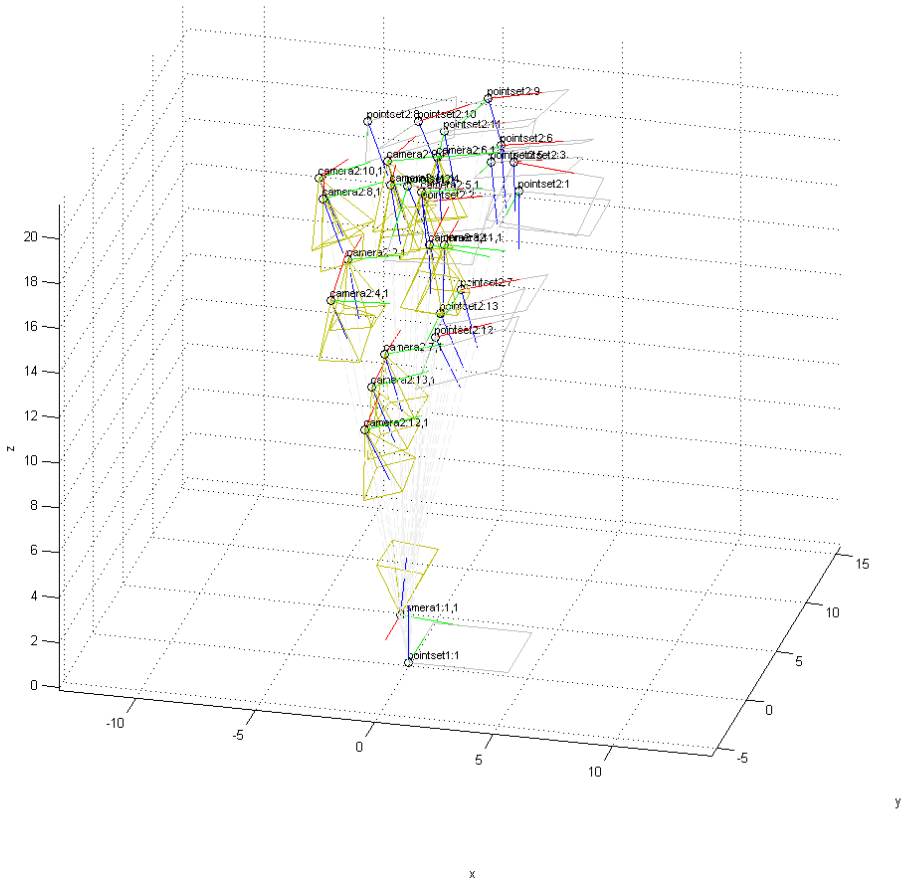


Figure 2.26: A depiction of the camera and calibration target frames for each observation for the facing smartphones experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

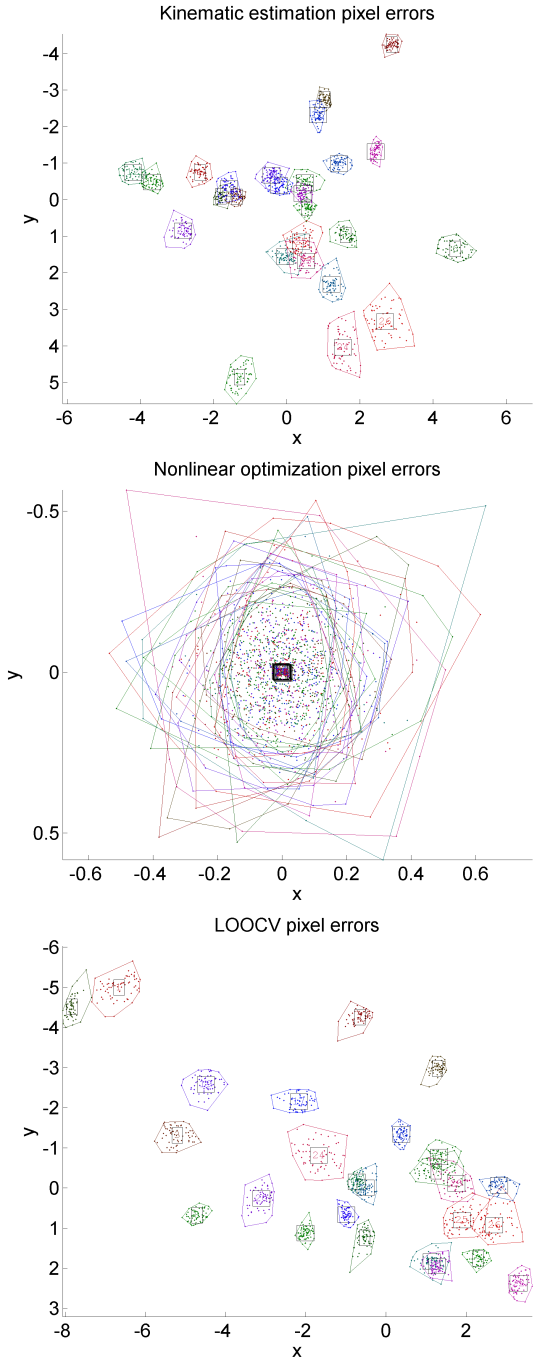


Figure 2.27: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the facing smartphones experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

2.10.6 Ad-hoc cluster

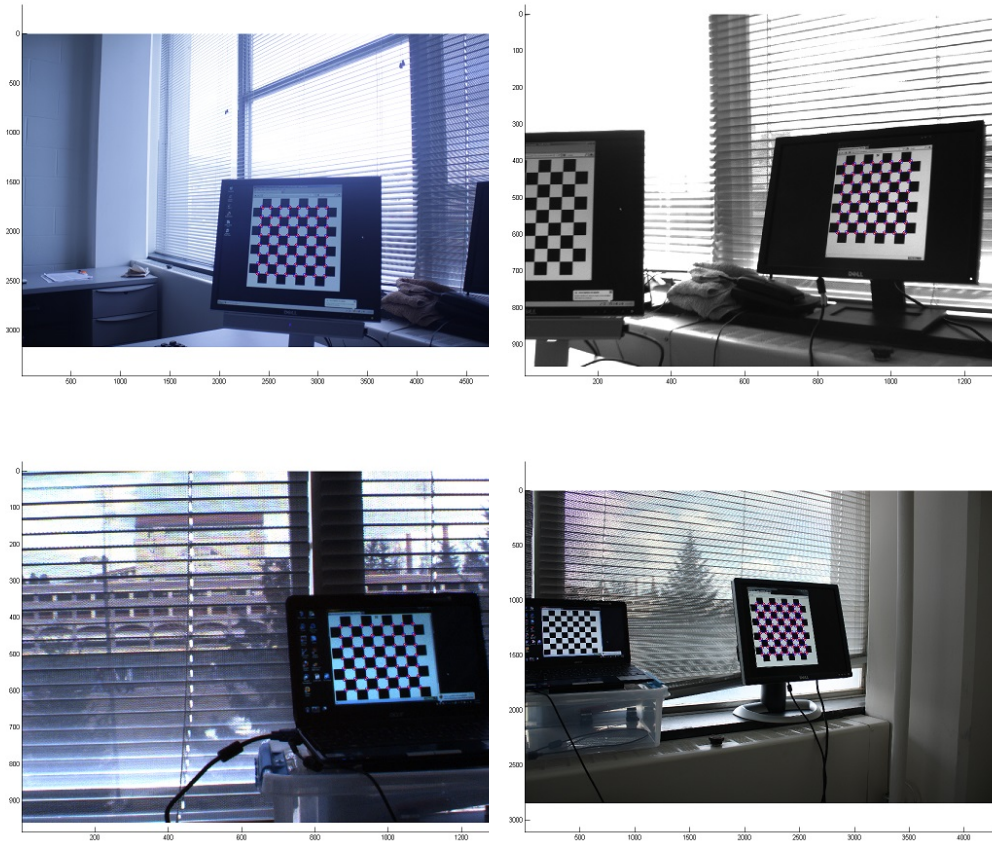


Figure 2.28: Examples of reprojections for the ad-hoc cluster experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. Although the fields of view of the cameras were not perfectly non-overlapping, no checkerboard's data was used for more than one camera.

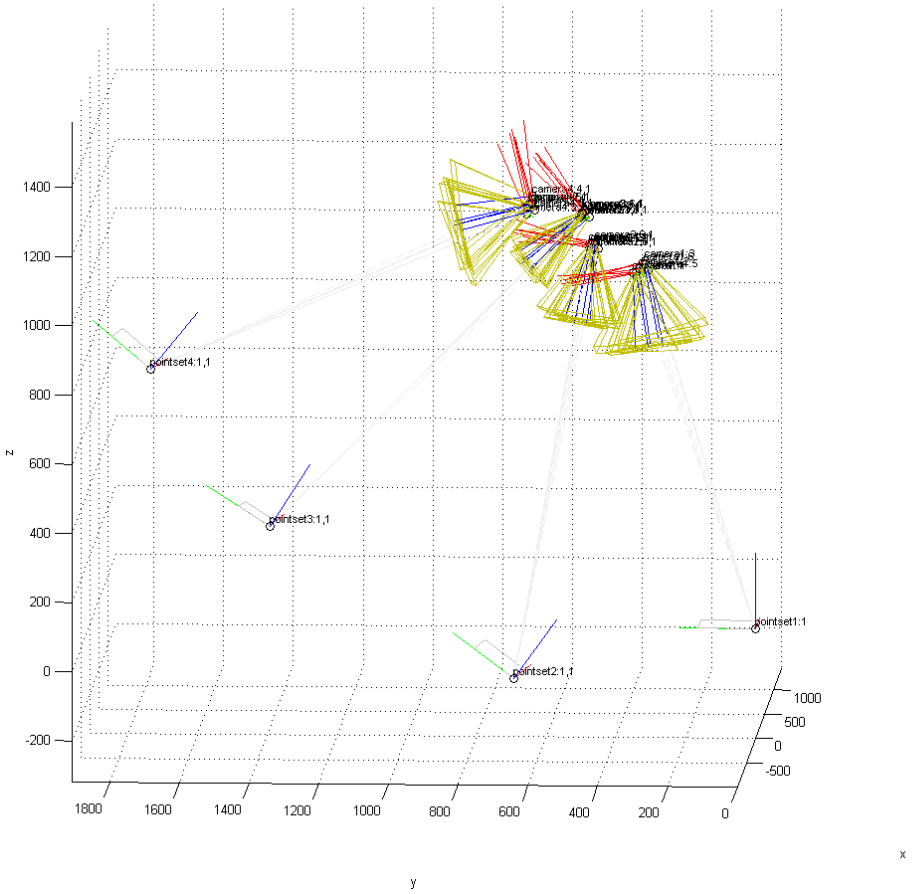


Figure 2.29: A depiction of the camera and calibration target frames for each observation for the ad-hoc cluster experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets.

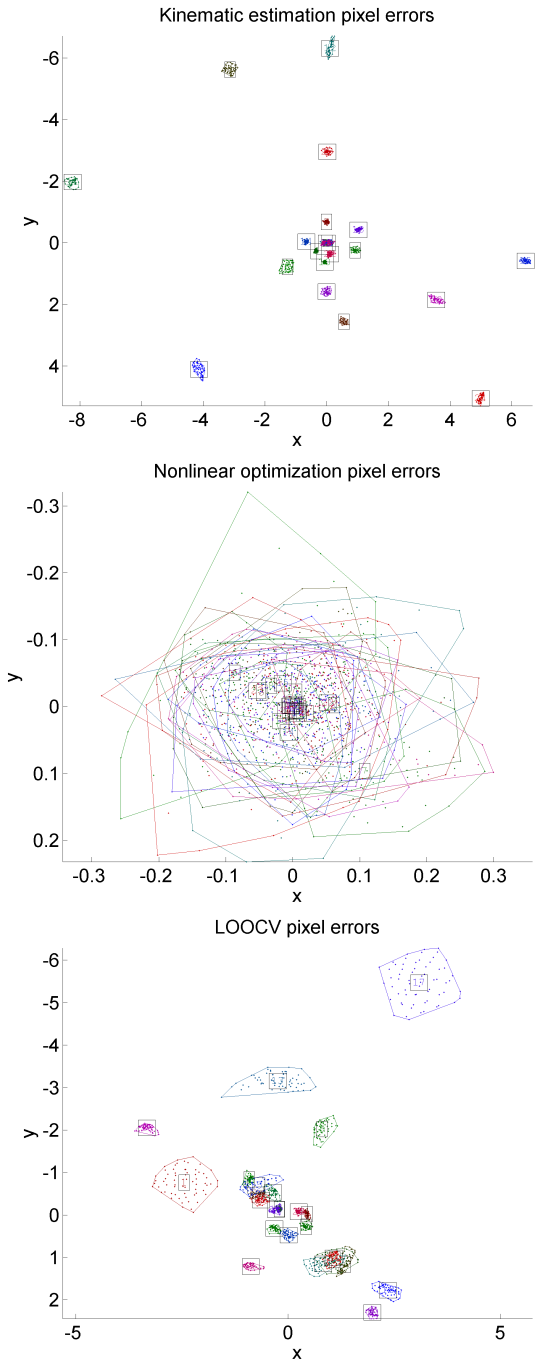


Figure 2.30: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the ad-hoc cluster experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

2.10.7 Point Grey Ladybug

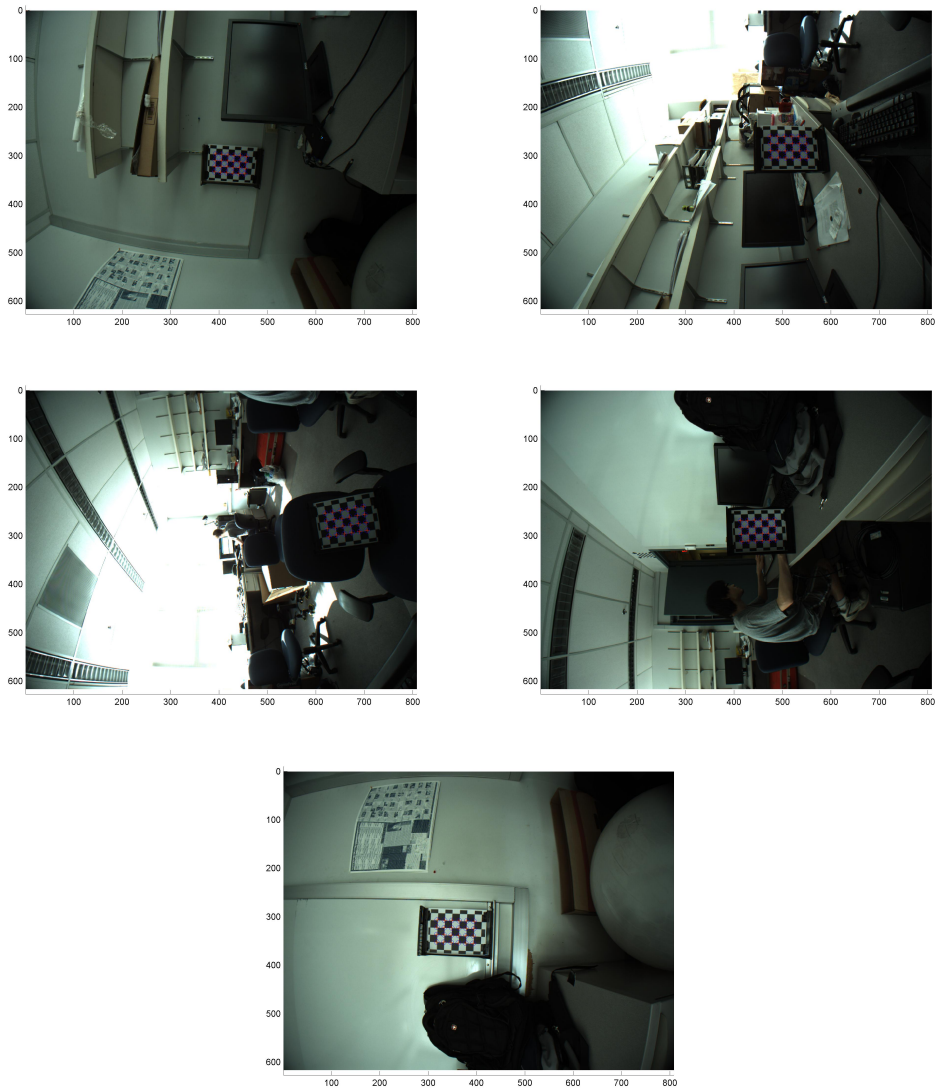


Figure 2.31: Examples of reprojections for the Ladybug experiment. The blue circles indicate observed checkerboard intersections. The red crosses indicate reprojections of the complete optimization. Although the frustra were not perfectly non-overlapping, no checkerboard's data was used for more than one camera.

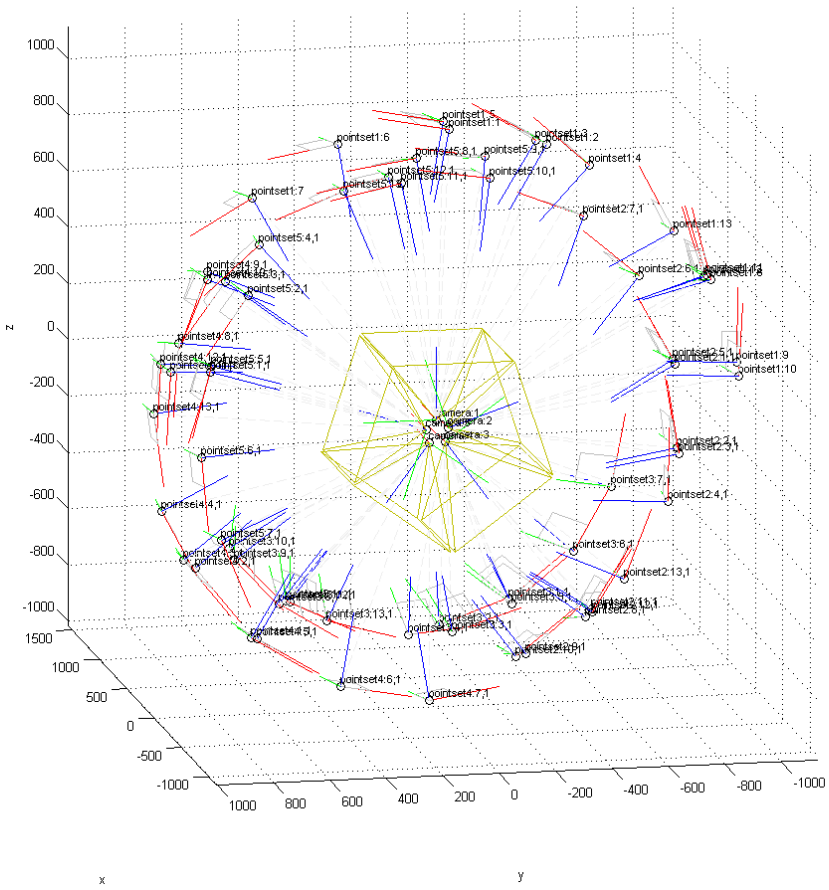


Figure 2.32: A depiction of the camera and calibration target frames for each observation for the Ladybug experiment. The x, y, and z axes are red, green, and blue respectively. Axes are labeled with the observation they came from. Origins are the camera center for cameras and one corner of the checkerboard for calibration targets. We have opted for a camera-centered view here.

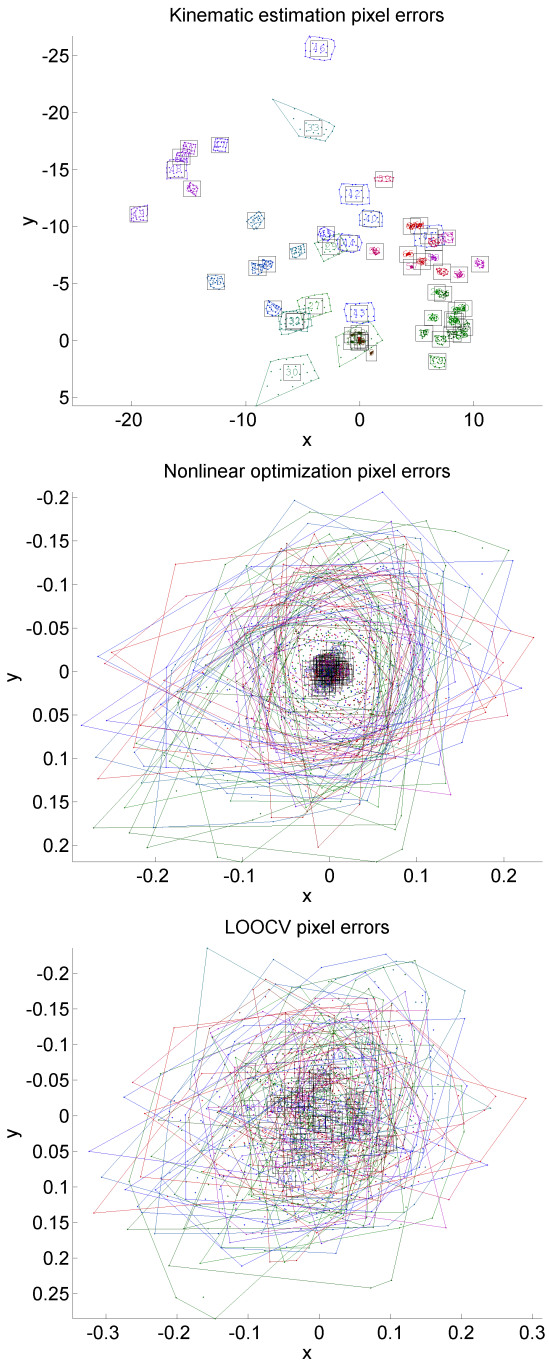


Figure 2.33: Pixel residuals after kinematic estimation (but before nonlinear optimization), solution after nonlinear optimization, and leave-one-out cross validation (LOOCV) for the Ladybug experiment. Colored points represent residuals for a single pixel of the correspondingly-colored observation. Colored polygons represent convex hulls for the residuals of the correspondingly-colored observation. There is one numbered box for each observation, at the average pixel residual for that observation. Note the change in scale between plots.

CHAPTER 3

SIMULATING THE STRUCTURE AND TEXTURE OF SOLID WOOD

Figure 3.1: Teaser: A guitar model featuring a couple different wood materials: quilted maple on the body and walnut on the neck. The image is rendered using our comprehensive, volumetric, procedural model of wood. We simulate most of the significant wood features: growth rings, pores, rays, and growth distortions. Furthermore, our model can produce the anisotropic specular highlight arising from reflection from the subsurface fiber structure, as seen in the quilted maple figure. The fiber directions are automatically derived from the growth distortions. Model by Nikos Natsios. **Inset:** A photograph of real quilted maple.

3.1 Introduction

Wood is a common decorative material in our surroundings, particularly in indoor scenes full of wooden floors and furniture, and also used in the design of many products (Figure 1). The conventional way to render wood is to use high-resolution 2D textures. These are commonly authored by using photographs to control the diffuse color, a microfacet model for the surface reflection, and a bump map derived from the color maps to introduce surface normal details. However, this approach has several shortcomings.

First, wood is fundamentally a volume phenomenon (see Figure 3.2). Texture maps derived directly from 2D photographs serve well for flat surfaces (including wood veneer), and these are easily transferable from one flat surface to another. However, curved and other more complex-shaped surfaces carved out of wood are difficult to photograph; the alternative of simply wrapping a flat texture around such a surface is often unsatisfactory. In addition to the inherent topological challenges in doing so, wood has distinctive, large-scale, volumetric structures such as growth rings whose surface patterns depend on the shape of that surface.

Furthermore, wood is composed of long, thin cells (fibers) that reflect light very anisotropically, which (with a clear smooth finish) causes a dramatic subsurface specular highlight. Previous work [55] has shown that the surface reflectance of finished wood can be represented well by a model that includes a diffuse component and a separately colored fiber-reflection component that is controlled by a direction texture giving the 3D direction of fibers at every point on the surface.

Procedural 3D textures for color variation in wood have been demonstrated before, and very realistic wood appearance has been demonstrated using color and

direction textures acquired using many images under controlled illumination, but both types of prior work have important limitations: previous procedural wood textures do not to justice to the beauty of the material, and measured textures are expensive in terms of time and equipment and cannot easily be transfered to models of different shape and size than the one measured.

This chapter aims to get the best of both worlds. The first contribution is to define a 3D procedural texture that produces realistic finished wood with all visually important features included. We present methods to simulate growth rings, fibers, pores and rays, including their effects on both diffuse and surface and subsurface specular components of the reflectance. These features are procedural, so they naturally adapt to any geometry or sawing plane, and they are modular, in that they admit a variety of possible methods for specification, which can easily be transferred, combined, and/or replaced.

Second, we demonstrate how to achieve realistic wood figure, i.e. anisotropic secondary highlights that vary in the patterns characteristic of particular highly prized types of wood. The core challenge is defining 3-dimensional *distortions* of the fiber structure, which result in fiber orientation fields that closely approximate their natural counterparts. This gives rise to results seen in Figure 1.

3.2 Related work

Our method builds directly on previous work in wood appearance and texture synthesis.

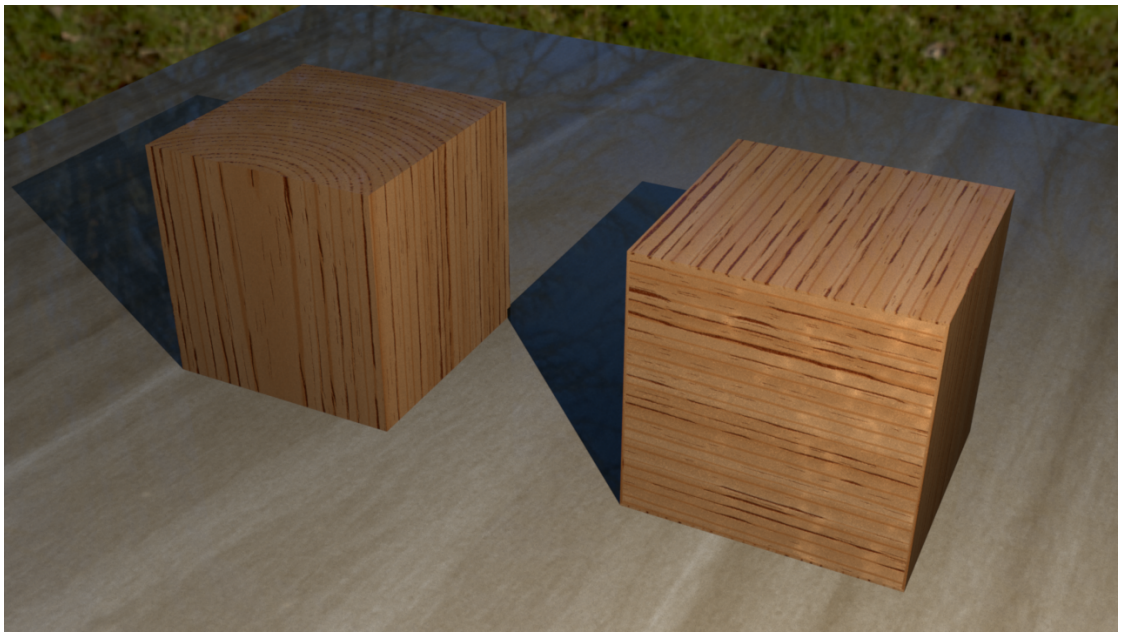


Figure 3.2: A rendered comparison of a carved wooden cube (left) and a veneered one (right). The carved block shows one face of each of the three major cutting planes: tangential on the left, radial on the right, and transverse on the top. The surface of the veneered block is made of six distinct thin slices and shows a tangential face on all sides. Clearly, it is substantially harder to produce the result on the left using 2D texturing techniques. Public domain environment map by GiantCowFilms.

Wood BRDF. The BRDF of our method is that of [55], with minimal changes and with added importance sampling. This BRDF is the sum of a surface reflection component, a diffuse component and a subsurface, anisotropic specular component. (In this chapter, by "specular" we mean glossy, not delta reflection.) For the subsurface specular component the BRDF models the reflection from subsurface fibers in much the same way reflection is modeled for hair and fur. The effect is that incoming light is scattered into a cone with some spread. Please refer to the original paper for the detailed definition and rationale behind the BRDF model.

This BRDF is able to capture effects seen in real wood that cannot be reproduced by surface-only anisotropy such as the Ward model [89]. However, acquiring parameter data has long been a practical problem in using this wood BRDF.

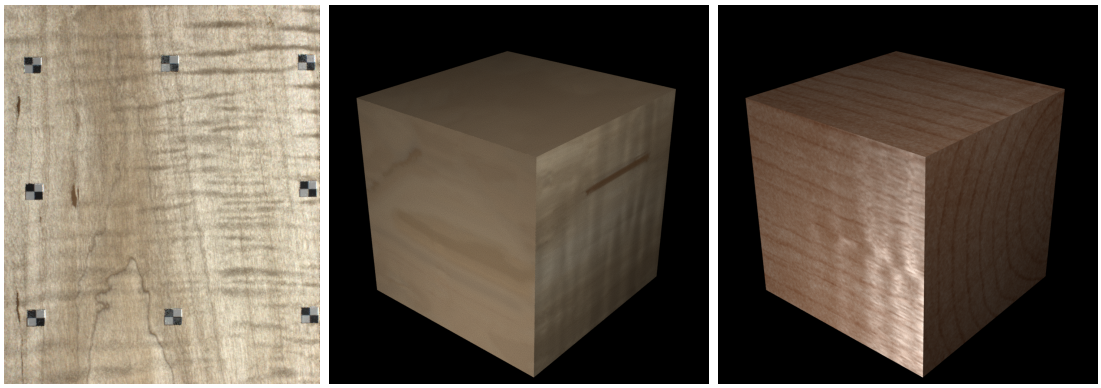


Figure 3.3: Generating solid texture by 2D-to-3D synthesis does not work well for wood. **Left:** Rendering produced using 2D data from [55]. **Center:** Rendering of the 128x128x128 solid texture set from [39], which was synthesized from a crop of the same 2D data. The results lack the global correlations found in real wood such as (approximately) cylindrical growth ring surfaces. **Right:** Rendering from our model, voxelized to the same texture resolution.

Marschner et al. [55] use a specialized fitting process which requires specific equipment and a large number of photographs per sample, resulting in small amount of 2D texture. In contrast, our volumetric method allows arbitrarily large synthesized texture and does not require 2-dimensional UV-mapping. (Note, however, that we do not use volume rendering, but simply "carve" the parameters for the BRDF out of the volumetric model.)

Solid textures. Solid textures [62] are a natural fit for wood, as many features of wood, such as growth rings, are fundamentally solid phenomena. Using solid textures also obviates the need for explicit surface parameterization. However, solid textures generally have to be defined procedurally, and previous procedural models for wood, often seen as demos or examples in rendering systems, have not produced very realistic results. A 3D voxelized representation could produce higher quality, but it is not clear where the data would come from, and storing the volume naively would be prohibitively expensive.

Noise functions. A key component of most procedural textures is noise func-

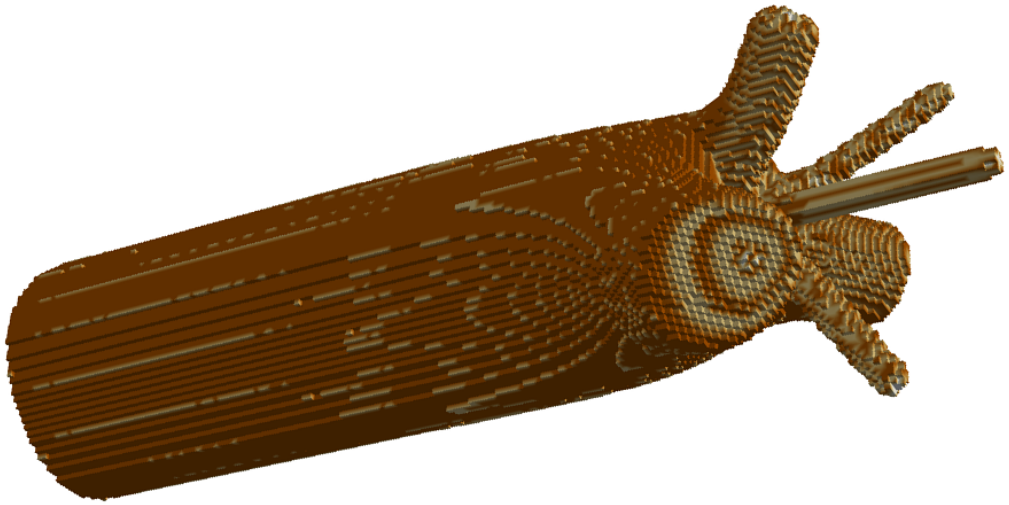


Figure 3.4: An example of progressive growth modeling, generated using the level set model of [69] coupled to a phloem transport model [68], both works by Sellier. Such models are able to model large-scale topological features such as branches and knots but are comparatively computationally expensive per resolution. Data provided by Sellier.

tions. A survey is given in [44]. Excessive regularity in a texture looks unnatural and jarring, and introducing (pseudo-)randomness helps produce pleasing variation in the resulting texture. Modeling wood as a solid texture consisting of cylindrical growth rings distorted by a noise function is a well-known practical technique [51, 10, 47]. However, previous techniques have stopped at approximating the diffuse color of wood growth rings.

Perhaps the most famous example of noise is Perlin noise [64, 65]. Other types of noise include cellular noise [91] and wavelet noise [17]. There is also the family of sparse convolution noises, which convolve a set of impulses with a kernel, first introduced by ([50, 51]) and further developed by [87]. More recently, the Gabor kernel has emerged as an attractive choice [45, 46, 23] due to its excellent spatial and spectral properties. In addition to these "continuous" noise functions, it is also possible to generate a field of scattered impulses in order to place objects. It is easy to use a white noise (uniform) distribution of impulses; for our purposes we

have found this to be sufficient. If less clumping is desired, one may consider blue noise distributions such as [42, 43]. We have tested most of these noises and found they work well in breaking symmetry of a naive, perfect wood model. However, these generic noise functions have difficulty adequately representing the distinctive shape of various types of figure seen in real wood, such as the quilted maple shown in the teaser (Figure 3.1).

Exemplar-based methods. Exemplar-based methods (see [90] for a general survey) have been successful in generating a larger amount of texture from a single exemplar and are applicable to a wide variety of exemplars, though they generally struggle at reproducing large-scale coherent structures such as tree rings without some form of supervision. Works such as [39] are even capable of generating a 3D texture from 2D exemplars. Figure 3.3 shows an example generated using [39]. Unfortunately, the results do not respect the structure of the material, and the appearance is not satisfactory. Furthermore, as mentioned before, materializing a high-resolution 3D texture over a full regular Cartesian voxel grid is too expensive for many applications. We instead only ever materialize 2D textures, which we use to produce a final 3D texture procedurally. Volume effects are captured by noise and by the interaction between components rather than by full 3D materialization.

Progressive growth modeling. Another approach is to progressively simulate the growth of the tree over time. Examples include voxel-based approaches [10], level-set methods [69], and L-systems [83]. Since these methods have greater access to global information, they are able to model large-scale topological features such as branches and knots as shown in Figure 3.4. However, they are very computationally expensive per resolution compared to random-access methods as defined in [48], which can compute the texture at any point in a constant amount of time

regardless of the coordinates of the point or what parts of the texture have been previous computed. Our method is closer to the latter category, though we hope to be able to handle some of these large-scale topological features in the future.

3.3 Wood anatomy background

Our model is organized around the anatomy of wood, so we begin with some discussion of the relevant features (see Figure 3.6 for an illustration of the planes used to describe these features: transverse, radial and tangential). We will focus on hardwoods (trees belonging to the angiosperms; not necessarily harder wood) because they contain a greater variety of anatomical structures than softwoods and also contain nearly all woods that are prized for their appearance. Softwoods generally contain a subset of the same features and can also be handled by our model, but other woody plants used for lumber (such as bamboo) are entirely different in structure and are not within the scope of this chapter [30, 60].

3.3.1 Seasonal growth

The most obvious feature in the appearance of most woods is growth rings. Growth rings result from the contrast between earlywood, which is produced during the spring, grows quickly, tends to be lighter in color, has larger and thinner-walled cells, and sometimes has larger pores (see below); versus latewood, which is produced during the summer and is the opposite. There are some species with indistinct growth rings, including many tropical hardwoods that grow in climates without strong seasonal variations.

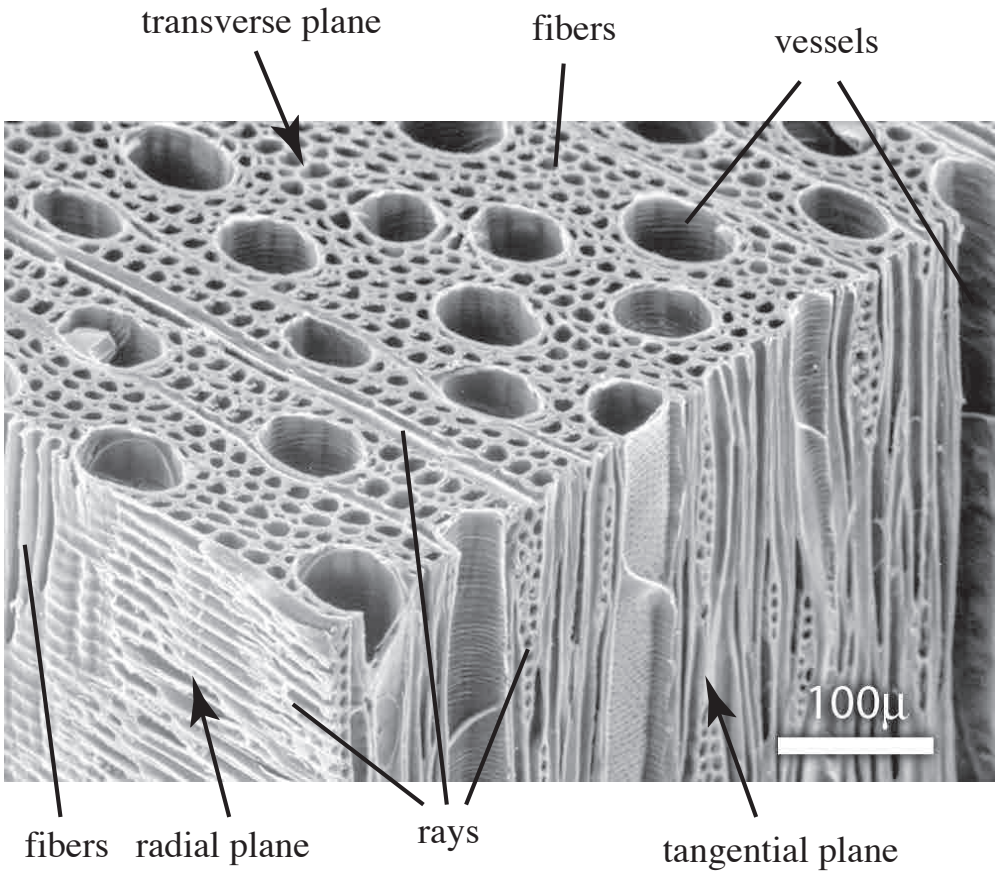


Figure 3.5: Scanning electron micrograph of red maple, showing vessels, rays, and fibers, as well as the radial, tangential, and transverse planes. [NC Brown Center for Ultrastructure Studies, SUNY College of Environmental Science and Forestry, Syracuse, NY]

3.3.2 Longitudinal and ray fibers

The majority of cells in wood (about 90% by volume depending on species) run in the longitudinal direction. However, there are also ribbon-like cells or clusters of cells which grow outward in the radial direction, called rays. Rays are typically very narrow in the circumferential direction and wider in the longitudinal direction. Depending on species, they may be too small to be seen easily by the naked eye or up to as large as about a millimeter thick and a few centimeters tall [30].

Rays can produce a striking visual effect since their fibers run in a direction

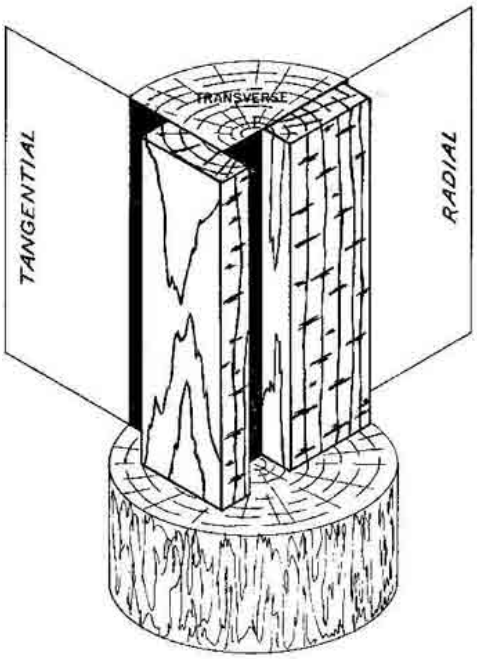


Figure 3.6: The principal cutting planes of wood. Diagram courtesy of Alabama Agricultural Experiment Station [4].

perpendicular to the main mass of fibers, thus producing a different reflective effect on the surface. On the tangential plane rays generally appear dark, whereas on transverse and radial planes they can be very bright for certain illumination configurations.

3.3.3 Pores

Pores (vessels) are hollow longitudinal tubes only found in the hardwoods. When cut, they produce openings or indentations on the surface of the wood. Pores often show up as grooves on the surface of the wood, with the length of the grooves depending on the cut. Unless the pores are filled, such indentations will remain even after the wood is coated. If a staining finish is applied to the wood, the pores will tend to absorb more of the finish, giving them a darker color than the rest of

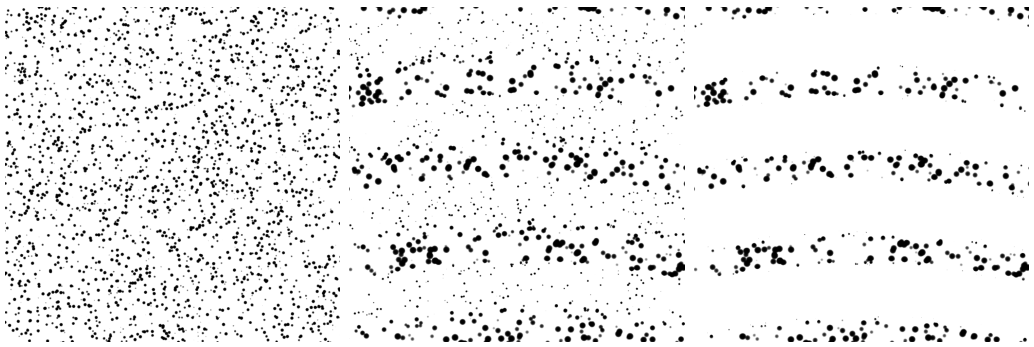


Figure 3.7: Vessel density may be uniform in diffuse-porous woods (left), only visible in the earlywood in ring-porous woods (right), or somewhere in-between in semi-ring-porous woods (center) as shown in these rendered diagrams.

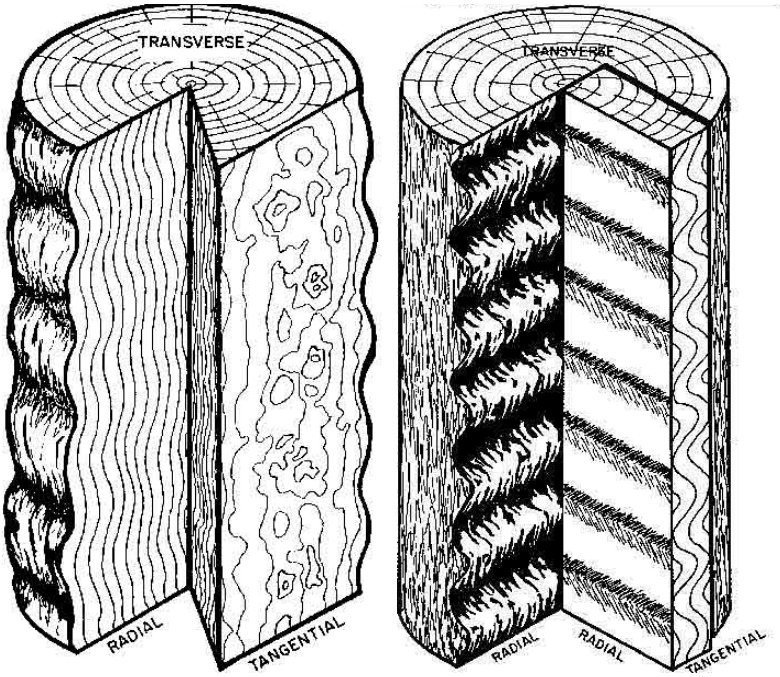


Figure 3.8: **Left:** Distortion in the radial direction produces variation in the shape of growth rings and is responsible for types of figure such as blister and quilted. **Right:** Distortion in the tangential direction does not affect the shape of growth rings, but is responsible for certain kinds of curly figure. Diagrams courtesy of Alabama Agricultural Experiment Station [4].

the wood.

The seasonal size and distribution of vessels depends on the species. In diffuse-porous species, the size of pores is independent of the season. At the opposite

extreme are ring-porous species, where large pores occupy most of the volume of the earlywood, but the latewood pores are negligibly tiny. In these species the pores rather than the inherent color of the wood may be the primary feature of growth rings. Still other species lie between these two extremes, with larger pores in the earlywood and smaller but still significant pores in the latewood. See Figure 3.7 for an illustration.

3.3.4 Figure

The default direction of the fibers in a tree is longitudinal (parallel to the tree axis) in what is termed straight grain. Variations in the fiber direction cause distinctive visual effects known as figure. Waves in the fiber can cause deviations in the radial and/or tangential directions (see Figure 3.8), which is termed curly or fiddleback figure. Bumps in the radial direction may also result in blister, quilted, or birds-eye figure depending on their shape [4]. With a clear finish, these features produce the prized secondary specular highlight patterns.

3.4 Simulating wood features

In this section, we provide an overview of all wood features modeled by our approach.

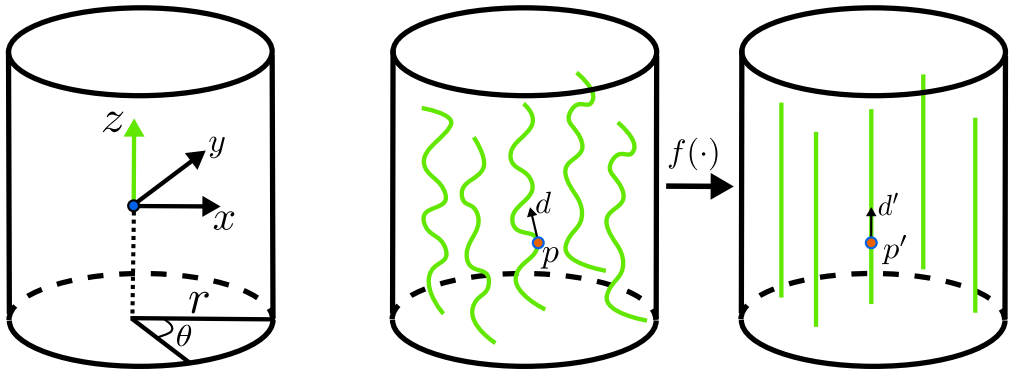


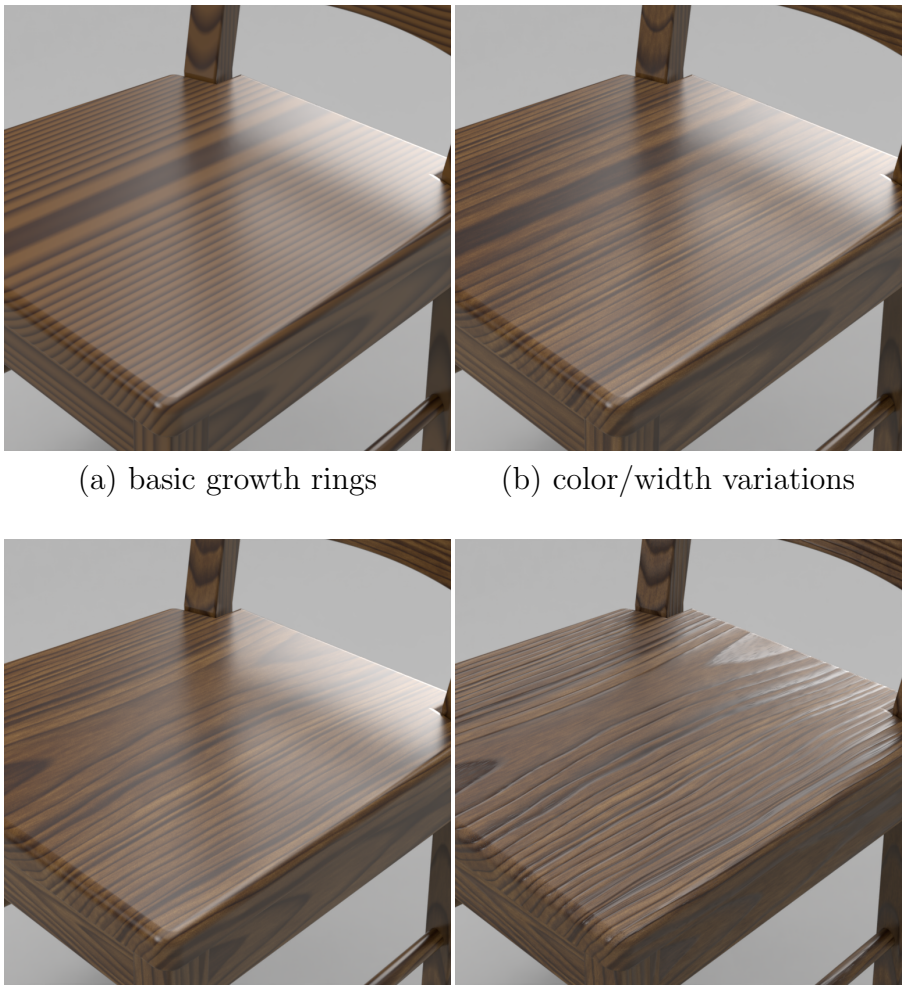
Figure 3.9: **Left:** an illustration of tree space. **Right:** We model distortion as a function $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$, mapping a position $\mathbf{p}$ in the distorted tree to a position $\mathbf{p}' = \mathbf{f}(\mathbf{p})$ in the undistorted tree. The distorted fiber direction $\mathbf{d}$ is mapped to the undistorted direction $\mathbf{d}' = J_{\mathbf{f}}\mathbf{d}$.

3.4.1 Coordinate systems

Below, we will assume all computation is happening in *tree space*: the z-axis of this space is aligned with the axis of the tree. We can parameterize the space using Cartesian coordinates x, y and z, or cylindrical coordinates r, θ and z. In our implementation, we assume the units of tree space are centimeters; this is important to maintain correct real-world scale for all wood features. An illustration of tree space can be found in Figure 3.9, left. The relationship between tree space and world space can be expressed as a 4×4 affine matrix transformation for every object in the scene with the wood material.

3.4.2 Basic growth rings

A core ingredient in the color of wood at a point $\mathbf{p}$ is the time at which the point was laid down during the growth process. In the simplest case, the color can be chosen as a binary decision between specified earlywood and latewood colors. We model this as a square wave as a function of radius r; the widths of the rings can



(a) basic growth rings (b) color/width variations

(c) distorted growth (d) pores/bump/roughness

Figure 3.10: Demonstrating the effect of the wood features modeled by our method.

be user-specified. However, the earlywood-latewood transition is often fuzzy, so we allow for controllable smoothing between the high and low values. The resulting simple growth rings are shown in Figure 3.10 (a).

3.4.3 Diffuse and specular color

While we can specify earlywood and latewood colors independently, we hypothesize that much of the diffuse color variation in wood can be explained by varying levels

of concentration of the same absorptive pigment(s). Therefore, we assume that all colors are drawn from a single Beer's law curve $c^{\alpha(\mathbf{p})}$. The base color c (for example, the earlywood color) represents the color of the absorptive pigment(s), while $\alpha(\mathbf{p}) \geq 0$ is proportional to the optical depth, with a higher value representing a greater concentration of pigments.

Given a single photograph of a target wood, we can produce an estimate of the colors by taking the 25th and 75th percentiles of each channel and treating those as the earlywood and latewood colors; we can also find the exponent α that approximately turns earlywood into latewood color. These estimates can be further adjusted by the user if needed.

From here, α for any given point is determined by the sum of a few factors. First, the season of the wood determines the blend between earlywood and latewood α. To this, we add 1D noise based on radius on the scale of years (i.e. square wave wavelength) so that the growth color varies from year to year. Then we add 3D noise to provide additional variation.

Secondary highlight color (i.e. the RGB weight of the fiber reflection BRDF component) is computed from the diffuse color by raising to a power less than 1. Intuitively, this color follows the same patterns as the diffuse color, but is less saturated, because it corresponds to shorter light paths through the wood than the diffuse component. This intuition is confirmed by inspecting the measured data of Marschner et al. [55].

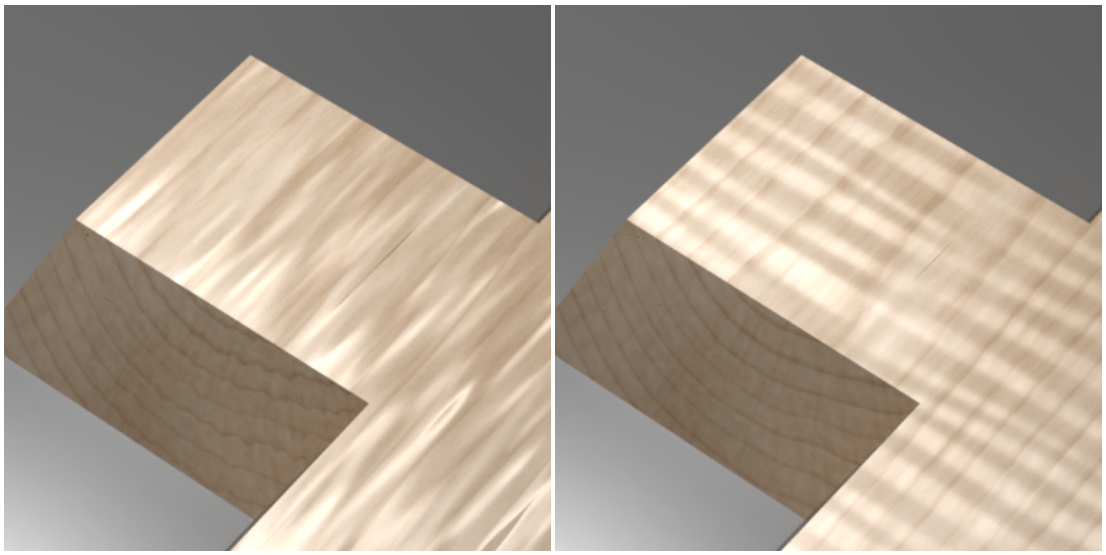
3.4.4 Color and ring width modulation

Furthermore, we use a 1-dimensional Perlin noise to modulate the earlywood and latewood colors of different growth rings, and a 3D anisotropic Perlin noise, providing small-scale color detail. The modulation is achieved through modulating the power α, as opposed to the color values directly. We also allow random variation in the growth ring sizes, by modulating with a 1-dimensional Perlin noise function of r. The effect of these additional modulations can be seen in Figure 3.10 (b).

3.4.5 Distortion and fiber direction

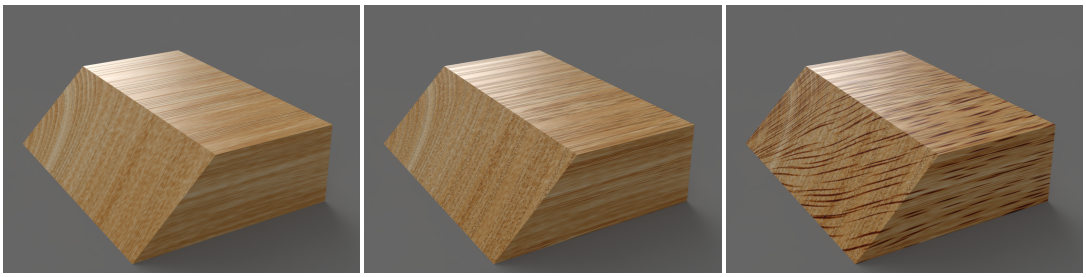
Real wood departs from the idealized cylindrical shape assumed above, both in terms of distinctive figure and less distinct random variations. To model these, instead of looking up points $\mathbf{p}$ directly, we apply a differentiable distortion field $\mathbf{f} : \mathbb{R}^3 \to \mathbb{R}^3$ to all lookups before querying the idealized (undistorted) tree. In other words, $\mathbf{f}$ is a mapping from the distorted into the undistorted tree; this can be seen as the inverse of the intuition of building an idealized tree model and finally distorting it. This is illustrated in Figure 3.9, right.

To define the distortion $\mathbf{f}$, we can use a generic noise model such as Perlin noise. This works well to break the unnatural symmetry and perfection of the undistorted tree; compare Figure 3.10 (b) and (c). However, to achieve a natural secondary highlight, we found that Perlin noise does not give satisfying results (Figure 3.11), and neither do other constructions like Gabor or impulse noise. Instead, we synthesize 3D distortions from 2D image examples; this is the most involved part of our model, explored in the next section.



Perlin distortion synthesized from 2D exemplars

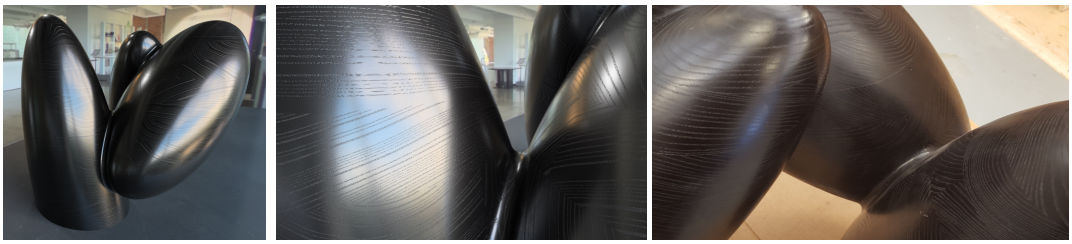
Figure 3.11: Our wood model produces secondary highlights through fiber direction variations, which are determined by the wood distortion function $\mathbf{f}$. We observe that Perlin noise (left) does not lead to natural curly maple highlight appearance. We instead synthesize a more involved 3D distortion from 2D exemplars (right); we give more details in Section 3.5.



(a) no pores and rays (b) with pores added (c) with rays added

Figure 3.12: Illustrating the effect of pores and rays on a small block of oak.

Distortion of vector fields. To distort a vector field, it is not sufficient to merely modify the lookup into an undistorted vector field $\vec{\mathbf{u}}$ by replacing $\vec{\mathbf{u}}(\mathbf{p})$ with $\vec{\mathbf{u}}(\mathbf{f}(\mathbf{p}))$. For example, merely changing the lookups into the fiber direction field does not result in any change since the undistorted fiber direction is equal to $(0, 0, 1)^T$ everywhere. Instead, we need to use the Jacobian of the distortion function. Since $\mathbf{f}$ maps the distorted tree to the undistorted tree, we observe that $J_{\mathbf{f}}$ maps the distorted vector $\vec{\mathbf{v}}(\mathbf{p})$ to the undistorted vector $\vec{\mathbf{u}}(\mathbf{f}(\mathbf{p}))$. For



Painted ash wood sculpture rendering Photo reference

Figure 3.13: A sculpture made of painted ash wood, illustrating a common ring-porous appearance of this species. The paint obscures the difference between earlywood and latewood color, but the surface normal variations due to pores still reveal the growth rings. Sculpture by Wendell Castle. Photograph and model of sculpture used with permission from Carl Bass.

nonsingular $J_{\mathbf{f}}$, this can be inverted as $\vec{\mathbf{v}} = J_{\mathbf{f}}^{-1}\vec{\mathbf{u}}$.

Fiber direction. Using the above, we can find the distorted fiber direction by multiplying the inverse Jacobian $J_{\mathbf{f}}^{-1}$ by the undistorted fiber direction, which is normally $(0, 0, 1)^T$.

Jacobian computation. We found that central finite differences work well in approximating the Jacobian, while requiring 6 evaluations of $\mathbf{f}$. However, we can also compute it analytically, for distortions of the common form

$$\mathbf{f}(\mathbf{p}) = \mathbf{p} + m_r(\mathbf{p})\vec{\mathbf{r}}(\mathbf{p}) + m_\theta(\mathbf{p})\vec{\theta}(\mathbf{p}). \tag{3.1}$$

Here $\vec{\mathbf{r}}$ and $\vec{\theta}$ are the normalized radial and tangential directions at $\mathbf{p}$, and m_r and m_θ are scalar functions describing the magnitude of the distortion in the respective direction. (We will drop the $\mathbf{p}$ dependence for clarity from now on.) We restrict the distortion to be in the xy-plane, since the longitudinal direction is less interesting as it does not affect fiber directions. This allows us to design distortions that correspond qualitatively to the radial and tangential distortion seen in real wood, as in Figure 3.8. The Jacobian of the distortion function is then

$$J_{\mathbf{f}} = I + \vec{\mathbf{r}} \otimes \nabla m_r + \vec{\theta} \otimes \nabla m_\theta + \quad (\ J_{\mathbf{r}} m_r + J_\theta m_\theta) \tag{3.2}$$

where $\otimes$ denotes an outer (tensor) product. The parenthesized terms can be ignored in practice with no obvious visual impact. This is because $\vec{\mathbf{r}}$ and $\vec{\theta}$ change slowly except close to the tree axis, so $J_{\mathbf{r}}$ and J_θ have small values (inversely proportional to r).

Dealing with foldover. Note that with zero distortion, $\mathbf{f}$ is the identity function, so the determinant of $J_{\mathbf{f}}$ is 1. A conceptually subtle but visually obvious problem happens when the magnitude of the distortion is too large, and the determinant flips to negative, causing a discontinuity in distorted fiber direction. While we can make the distortion magnitudes m_r and m_θ small enough that this does not happen, it is not easy to guarantee this, especially with user adjustments being allowed. If we need such a guarantee, we can compress the sum of the radial and tangential terms from equation 3.2 to always be less than 1, thus modifying the equation to

$$J_{\mathbf{f}} \approx I + \frac{\vec{\mathbf{r}} \otimes \nabla m_r + \vec{\theta} \otimes \nabla m_\theta}{\sqrt{1 + \|\nabla m_r\|^2 + \|\nabla m_\theta\|^2}} \tag{3.3}$$

This ensures that $J_{\mathbf{f}}$ has positive determinant everywhere and thus never passes through a singularity. Meanwhile, small gradients are nearly unaffected. Also note that the distortion itself is not affected; just the Jacobians and the resulting fiber directions.

3.4.6 Pores and rays

We model *pores* (vessels) as straight lines through the undistorted tree; the distortions affect them like the rest of the wood. More precisely, we distribute the pore centers in the 2D cross-section of the tree using stratified random sampling with a user-defined cell size. We find the pore locations within a cell by hashing the cell

index using the Burtle hash; this makes the pore locations fully procedural and alleviates the need to store them.

The effect of pores on the appearance is twofold. First, they cause surface grooves, i.e. negative offsets on the surface heightfield, which accordingly modify the shading normals. (Note, we do not currently use displacement mapping, instead just modifying surface normals according to the partial derivatives of the surface heightfield.) Second, they often cause darkening, especially if stain has been applied to the wood. We achieve both effects by defining *pore weight*, the influence of the pore on a given point in the undistorted tree. The weight is a decreasing function of distance r_p from the pore. We use the Wyvill kernel $w(r_p) = \max((1 - r_p^2/r_m^2)^3, 0)$ [70, p. 387], where r_m is the distance at which the pore influence decays to zero. We then directly use this weight (scaled by a user-specified strength) to offset the heightfield and to increase the color exponent. The effect is shown in Figure 3.12 (b).

To represent *ring-porous woods*, we can vary the parameters per wood type, causing the pores to become much smaller and less dense in the latewood. This effect can be observed in both oak (Figure 3.12) and ash (Figure 3.13).

Modeling *rays* is similar to pores. Since rays run radially, we instead use stratified sampling in θ and z, but we reuse the idea of hashing and also use a Wyvill kernel to define the ray weight as a function of distance from the ray center (in Cartesian, not cylindrical space). However, here we make the weight kernel anisotropic, since we observe that rays are often not circular tubes, but resemble "ribbons" with an elliptical cross section, broader in the z-direction. We do not modify surface normals based on the ray weight. The effect of rays is shown in Figure 3.12 (c).

3.4.7 Additional effects

Below, we describe some additional simple effects, adding to the expressive power of our model.

Earlywood bump. In addition to the indentations caused by pores, in some species, earlywood is softer and absorbs more surface finish; this causes the surface heightfield to "sink" lower in earlywood regions. We model this effect by lowering the heightfield by scaling the same smoothed square wave we used to determine the colors.

Earlywood and pore roughness. We observe that the recessed surface areas (pores and earlywood rings) sometimes have higher specular roughness. We hypothesize that this is because the recessed areas are softer and absorb more polish into the wood, so less is available on the surface, or because prolonged use of a wooden object leads to less mechanical polishing in the recessed areas. This effect is easily modeled by using two user-specified roughness values for the recessed and non-recessed areas, with soft interpolation at the boundaries.

Paint and stain. We optionally allow to specify the earlywood and latewood colors manually (stain) or make them identical (paint). Figure 3.13 shows an example of painted ring-porous ash wood. Note how the ring effect is still important, even though the paint obscures the difference between earlywood and latewood color.

3.5 Defining distortions for figure

We have presented a decomposition of the structure and appearance of wood into simple features. Most of these features can be satisfactorily modeled using random functions, including ring width, color variation, and ray/pore placement. Random distortions also suffice to simulate the minor irregularities found in wood with straight grain, but the abnormal growth patterns discussed in Section 3.3.4, which lead to the beautiful figure seen on finished wood surfaces, have specific structure that is not amenable to generation from noise. In this section we discuss how we author these distortions—specifically m_r and m_θ in Equation 3.1—which are responsible for both growth ring shapes and figured highlight patterns. (Distortions in the z direction are not needed for common types of figure, so we omit m_z.)

Our approach to modeling figure starts with the observation that the patterns we need to model are generally functions of only two of the three polar coordinates. Curly or fiddleback figure, as in Figure 3.11, is produced by oscillations in the θ direction that vary quickly in the z direction and somewhat in the r direction, whereas blister, quilted, and bird's-eye figure are produced by distortions in the r or θ directions that vary predominantly with z and θ.

We seek to provide a direct means of defining these patterns so that they look correct when sliced along the relevant plane and remain consistent throughout the volume. We do this by using a one-channel 2D texture to define the displacement magnitude in the r or θ direction, i.e. m_r or m_θ of Equation 3.1.

In the case where the distortion varies predominantly in r and z, the texture respresents a function of r and z. The scheme is trivial: simply revolve a one-channel 2D texture around the z-axis, and interpret the value as a displacement

in either the r or θ direction as shown in Figure 3.14.

The case where the distortion varies predominantly in θ and z is more difficult: we must define a globally consistent distortion magnitude as a function of θ and z. Simply making distortion a function of these variables (i.e. wrapping the texture around a cylinder and projecting along the r direction) leads to severe stretching of the map as r changes. Our solution to this problem instead wraps the 2D texture onto an Archimedean scroll (again, shown in Figure 3.14) and is outlined in the following section.

Example images for the four combinations of these two cases along with displacement in the r or θ direction are shown in Figure 3.16. We also discuss how we can generate distortion textures by texture synthesis in Sections 3.5.2.

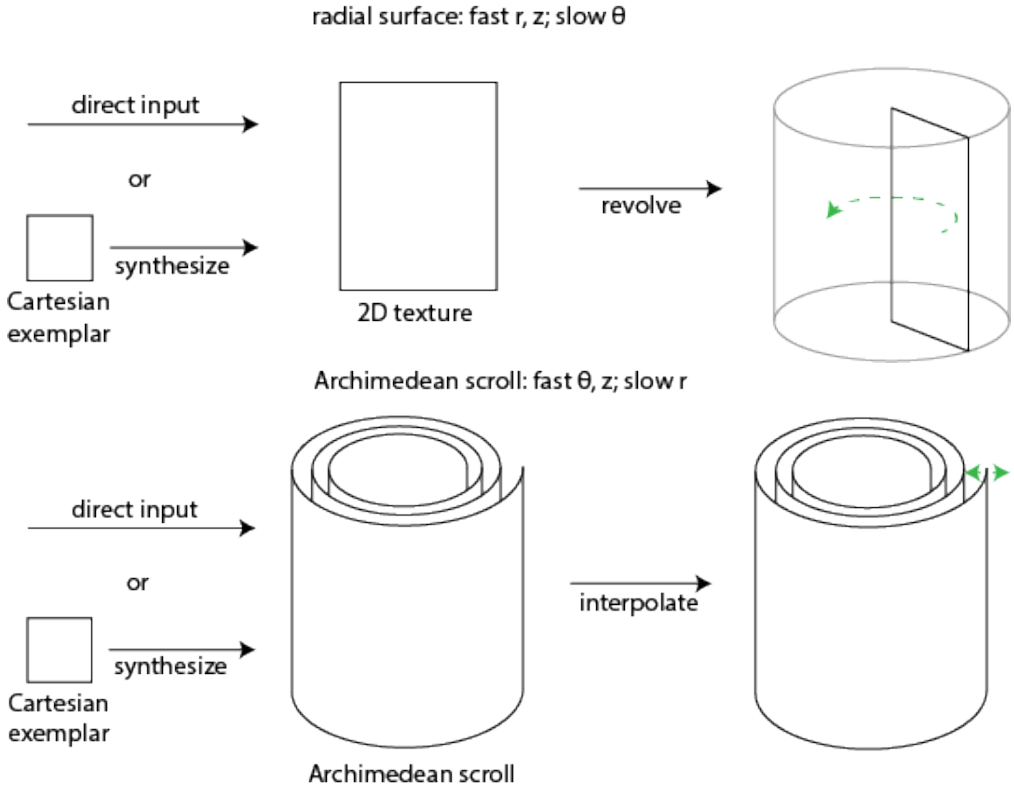


Figure 3.14: By assuming a cylindrical texture varies slowly along at least one cylindrical direction, we can represent it using only 2D information. These 2D textures can either be input directly, or synthesized using a small exemplar. The interpolation scheme for the spiral case is shown in Figure 3.17.

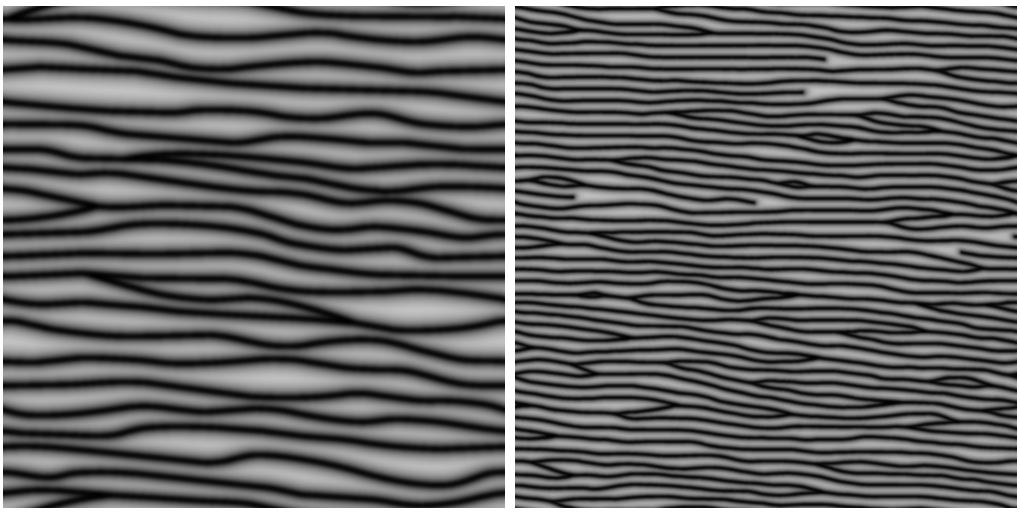


Figure 3.15: Given a small exemplar texture, we can use standard methods to synthesize a larger, tileable texture. This texture is then placed on either the radial plane or Archimedean scroll as per Figure 3.14.

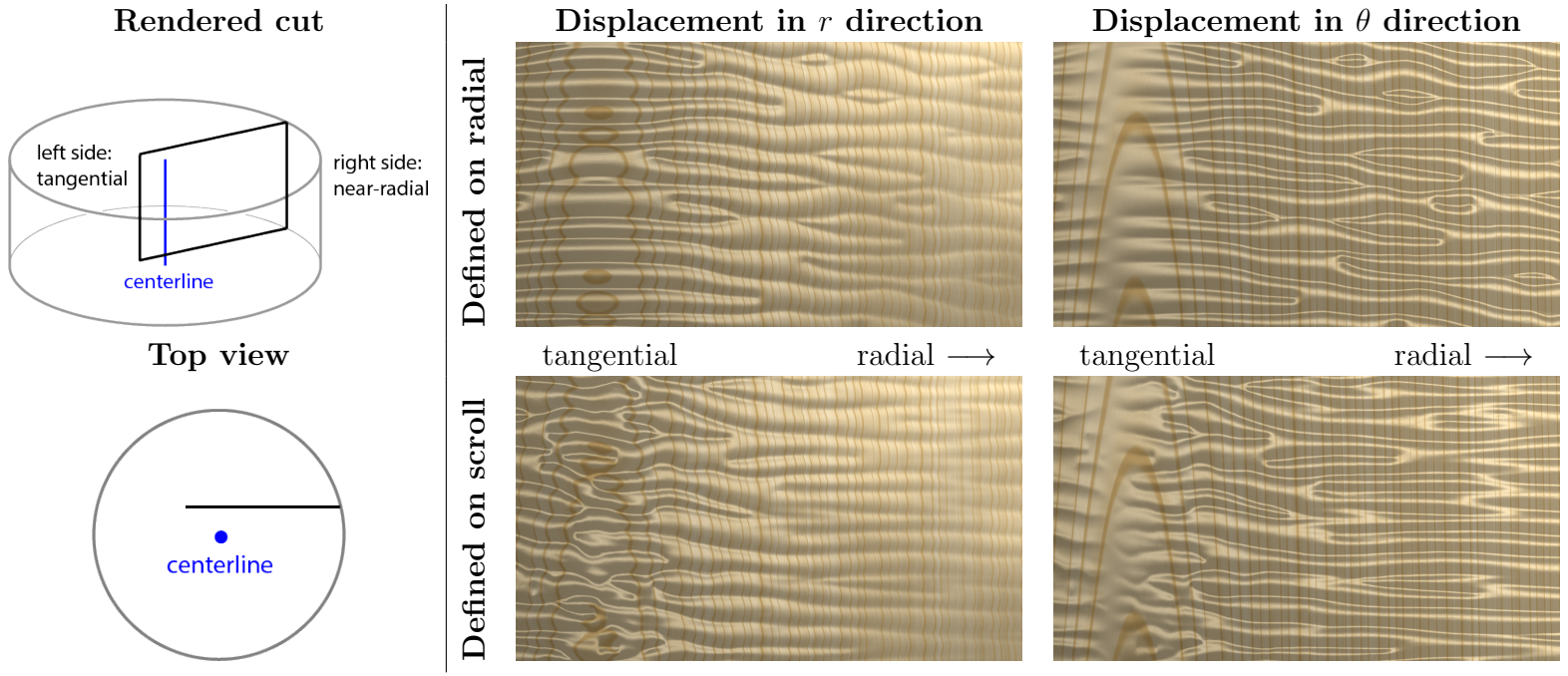


Figure 3.16: Example renderings of the four combinations of displacement in the r versus θ directions, and placing the input texture on the radial plane versus on the Archimedean spiral. The top-right and bottom-left images correspond to the cases shown in Figure 3.8. The rendered cut is tangential on the left side and near-radial on the right side. Figure is exaggerated and other features are simplified for exposition. See the supplemental material for videos of the four cases.

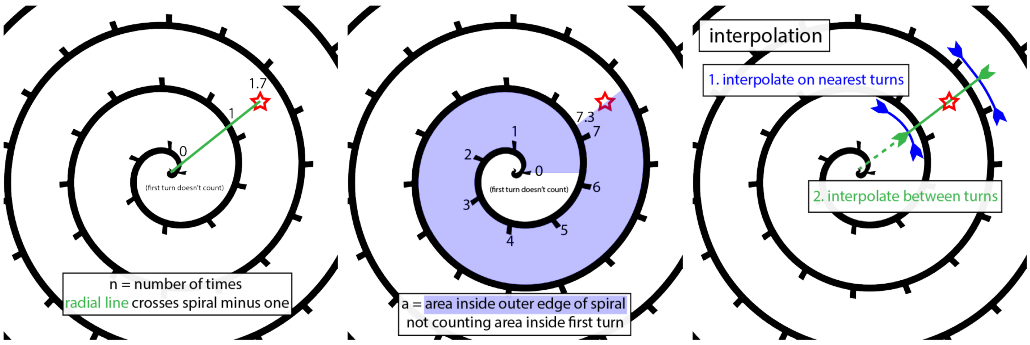


Figure 3.17: Archimedean spiral. **Left:** n is the number of times a radial line crosses the spiral minus one (linearly interpolated if a point lies between turns). **Center:** a is the area between the origin and the turn directly outside of a given point, not counting the area inside the innermost turn. Note that while texels lie on the spiral, they are indexed by the area within one turn further away. **Right:** Interpolation procedure. First, we interpolate within each of the nearest turns of the spiral. Then we interpolate between the results of each turn to get the value at the queried point.

3.5.1 Defining distortion as a function of z and θ

The goal of this section is to extend a 2D texture to a 3D texture, in such a way that slices of the 3D texture look like the input texture where the slicing plane is nearly tangential. In order to maintain the right look across a range of different positions, we wish to maintain constant linear rather than angular feature sizes as we get further from the z-axis. But since the circumference increases with radius, this requires increasing the number of features as r increases. The texture therefore cannot be completely constant in the r direction.

One solution might be to interpolate between textures defined on a series of concentric cylinders, each with the texture defined at the appropriate scale, and then interpolate between the cylinders. However, under such a scheme it may not be possible to tile an arbitrarily-sized texture around a given cylinder, and the uneven number of elements around each cylinder complicates addressing.

Instead, we elect to use an Archimedean scroll (an extruded Archimedean spiral), which has the advantage of unrolling into a single plane, greatly simplifying addressing and tiling. See Figure 3.17 for diagrams of the spiral and the grid, offset, and interpolation scheme described in the rest of this section.

Specifically, let t be the total angle of the spiral starting from the center; we parameterize the spiral by $s = \frac{t}{2\pi} - 1 \in [-1, \infty)$, the number of turns of the spiral not counting the innermost turn. Let r_1 be the spacing between turns—typically much larger than the size of a texel. The equation of the spiral is

$$\frac{r}{r_1} = 2\pi t = s + 1 \tag{3.4}$$

To actually index a point into the spiral, we use two parameters n and a.

The radial parameter n is equal to one less than the number of turns of the spiral crossed by a line between a the origin and a given point; n is real valued and increases linearly along radii between turns. For a point (r, θ) in polar coordinates, we have

$$n = \begin{cases} \frac{r}{r_1} - \frac{\theta}{2\pi} & \text{if } \frac{r}{r_1} \geq \frac{\theta}{2\pi} \\ \frac{2\pi}{\theta}\frac{r}{r_1} - 1 & \text{otherwise} \end{cases} \tag{3.5}$$

where inside the innermost turn we interpolate linearly from -1 at the origin to 0 at the innermost turn of the spiral.

After computing n, we can determine the s of the point on the spiral directly inside (r, θ) by

$$s = \lfloor n \rfloor + \frac{\theta}{2\pi} \tag{3.6}$$

From here, we compute a, the circumferential parameter, which is also real-valued. This is equal to the area between the origin and the turn directly *outside* (r, θ),

not counting the area $\pi/3$ inside the innermost turn. We choose to use the area instead of arc length, as the area has a simpler expression and the difference is negligible when not near the origin.

$$a = \begin{cases} \pi\left(s^2 + s\right) & \text{if } s \geq 0 \\ \frac{\pi}{3}\left(s^2 + 2s\right) & \text{otherwise} \end{cases} \tag{3.7}$$

Texel centers lie on the spiral, but they are indexed by the area within one turn further away (see Figure 3.17). In practice, to avoid sharp changes at the origin, we fix all texel values for $a < 0$ to be the same as $a = 0$.

To interpolate a texture, we take the nearest points on the spiral on each of the nearest turns (e.g. s and $s + 1$ for linear interpolation), compute a for each nearest point, interpolate within each turn using the fractional part of a, and then interpolate the resulting values between turns using the fractional part of n.

In addition to texturing, these can also be used to define a grid and offsets in order to produce Perlin, sparse convolution, or other forms of noise, with anisotropy naturally aligned with polar coordinates.

3.5.2 Synthesis

In addition to direct authoring of the entire texture, it is convenient to be able to create tileable textures, and to synthesize a larger texture from a smaller one. The radial surface is already a 2D Cartesian surface, and the Archimedean spiral can be unwrapped into one. Synthesis of larger and/or tileable textures can therefore done effectively for these using standard 2D-to-2D texture synthesis methods such as those given in [49]. Since the viewer typically only sees a 2D slice through the final 3D texture, and many different features participate in the final result, naive

tiling is not as obvious as it would be on a 2D surface; however, if less regularity is desired, on-the-fly texture synthesis (again as given in [49]) or Wang tiling [16] could easily be used.

3.5.3 Appearance of figure

Figures 3.1 and 3.16 show the effects of distortions on the appearance of the cut surface. Variation in the distortion shows in the color patterns and also causes changes in the fiber directions that dramatically change the appearance of the subsurface specular highlight. Growth ring shapes are affected only by distortions in the r direction, since distortions in θ only move points parallel to the rings. For the fiber highlight, the most dramatic effects are caused by distortions perpendicular to the surface, since these cause the fiber directions to swing out of the plane. Distortions parallel to the surface (seen near the right side of each image in the r column and near the left side of each image in the θ column) only rotate the fiber direction in the surface, producing figure that is only noticeable when the illumination is far from the normal. We used two distortion maps to produce figure in this chapter, one texture (Figure 3.15) containing near-parallel ripples to produce curly, or fiddleback, figure and one with more irregular sharp-bottomed valleys separated by round-topped hills to produce quilted figure.

3.6 Additional results

We implemented our model in the Mitsuba renderer [33], and also in Autodesk Fusion 360, a commercial modeling and rendering system. To evaluate rendering

performance of our method, we rendered the teaser image (Figure 1) on an Intel Xeon E5-2650 v2 machine (16 cores, 32 threads), which took 310 seconds. We then re-rendered the same image, but replacing all wood materials with a simple white plastic; this took 247 seconds. The rendering used a standard path-tracer with 512 samples per pixel and an NL-means denoising pass. The algorithm runs on-the-fly during rendering, so there is no precomputation time. After becoming familiar with the parameters, an experienced designer can create a preset in about ten minutes.

Matching renderings to photographs. An important question is how to set the parameters of our model to match physical wood samples or their photographs. We created a GUI previewer for setting the parameters of the wood model, shown in Figure 3.18. Using this tool, combined with some skill and experience, it is possible to obtain close matches between photographs and renderings of many species, as seen in Figure 3.19. We also show a recording of this process in a supplementary video. Automating this process is an interesting and difficult problem left as future work.

Wooden floor. One common real-world application of wood is flooring, ranging from simple parallel board patterns to elaborate parquetry. A nested-square pattern is shown in the Sponza scene in Figure 3.20. A board pattern can be defined by a different world-to-tree affine transformation for each board; we define these by randomization.

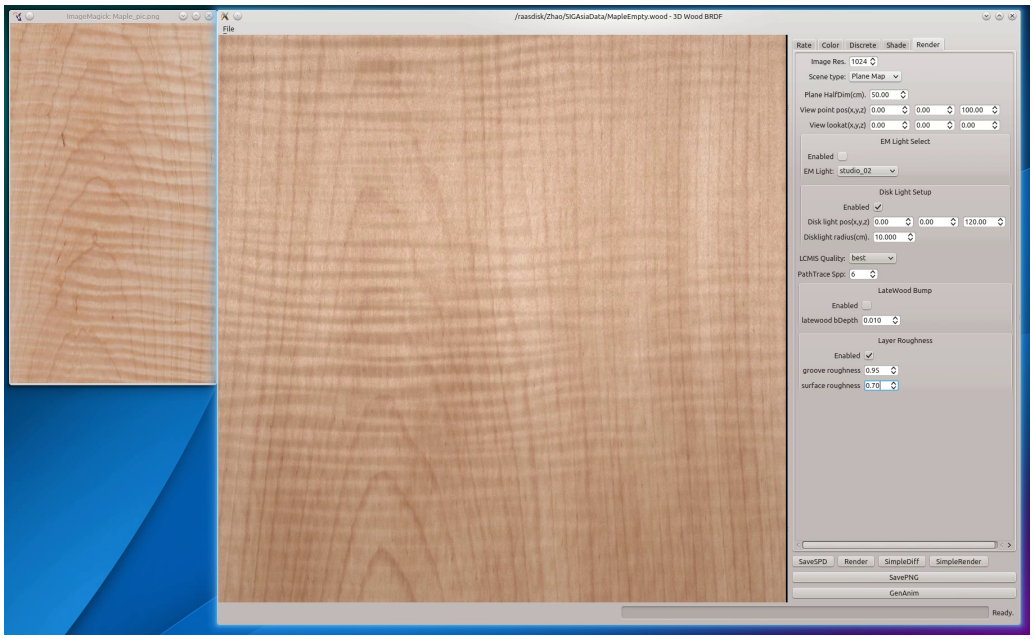


Figure 3.18: A screenshot of our wood previewer. On the left is a photograph of real wood, on the right is our result. We also show a recording of the parameter-setting process in a supplementary video.

3.7 Conclusion and future work

In this chapter, we presented a comprehensive volumetric simulation of the appearance of wood, including components such as growth rings, pores, rays, and growth distortions. In addition, our model fully supports anisotropic specular figure caused by wood fibers, common in curly maple and other species. The fiber directions required for such secondary highlights can be derived directly from the growth distortions introduced by our model; these 3D distortions are generated from 2D exemplars using constructions designed to match the look of radial and tangential cuts. The components of our model are intuitive, easy to control, admit efficient computation and require minimal storage.

There are several possibilities for future work. We could add support for deviations from the cylindrical growth model assumed in this chapter. These include

Figure 3.19: Photo (left) vs. rendering (right) for different wood species, top to bottom: walnut, pine, ash and curly maple.

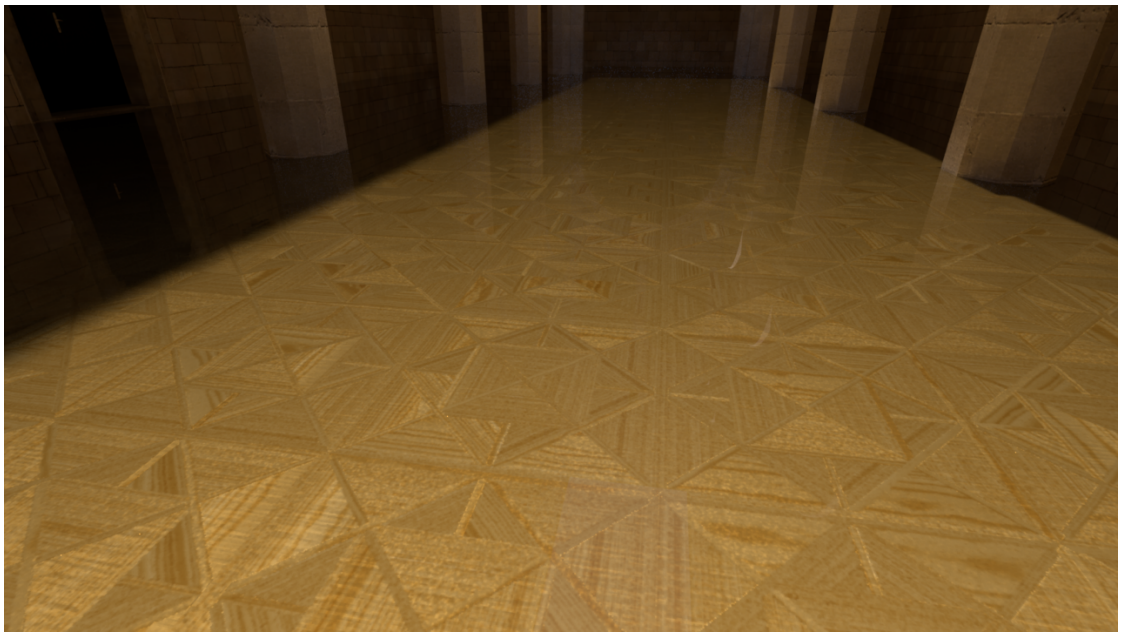


Figure 3.20: Nested-square parquetry floor pattern in the Sponza scene. Model by Marko Dabrovic, Kenzie Lamar, and Morgan McGuire; downloaded from Morgan McGuire's Computer Graphics Archive [57].

knots, which are parts of small branches embedded in the trunk of the tree; burls, areas of abnormal, chaotic growth; and crotch figure, which occurs when the trunk splits into two large branches. It may also be worth deriving or approximating the wood BRDF using a more principled analysis of multiple scattering through the wood fiber geometry. Finally, a more automatic way of setting the parameters and input textures of our model from photographs would be valuable.

CHAPTER 4

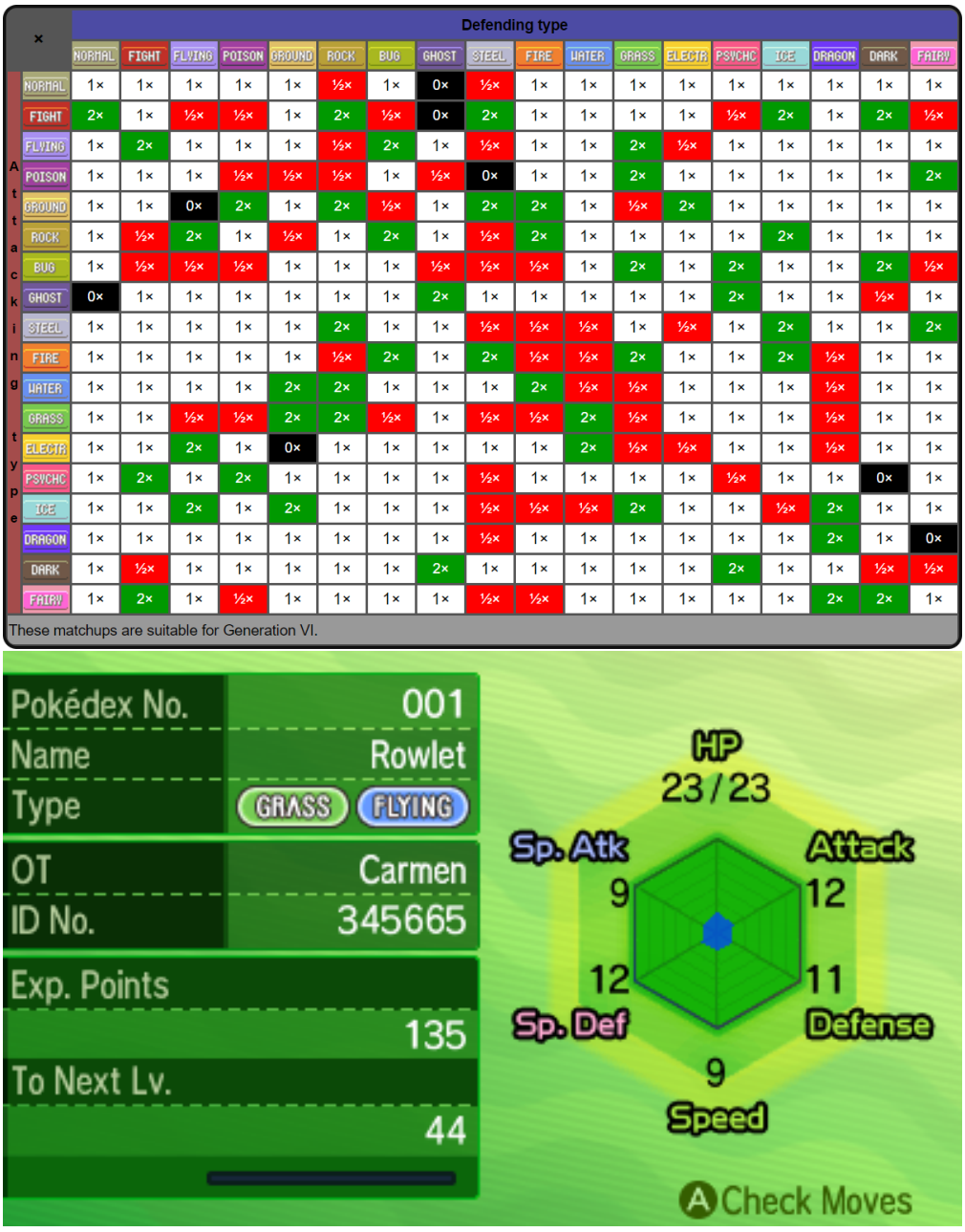
BALANCING ZERO-SUM GAMES WITH ONE VARIABLE PER STRATEGY

4.1 Introduction

"A game is a series of meaningful choices", as *Civilization* designer Sid Meier is famously quoted [67]. A choice is not very meaningful if it is the correct choice every time, or the wrong choice every time. Overpowered choices will tend to crowd out all others, leading to stale gameplay; underpowered choices may rarely see use in practice, wasting the development resources spent on them. Game designers thus strive to balance the choices in their games so that each can be reasonably employed.

While human playtesting is the ultimate arbiter of balance and player enjoyment, dedicated human playtesting is time- and labor-intensive. Analytics techniques [1, 2, 3, 52] can allow designers to measure the impact of decisions on balance and other design objectives using large-scale experiments with actual players. However, in this case the game's playtesters are also its audience and customers, and therefore expect to enjoy the game while they are playing it. Therefore designers cannot only be concerned with the final state of a game's balance, but also the speed at which balance is achieved and even the initial balance.

An alternative to human playtesting is to use AI players. AI techniques such as Monte Carlo Tree Search [14, 8] are capable of achieving a high level of skill and are adaptable to a wide variety of games, a recent high-profile example being AlphaGo [72]. AI players have also been used to generate and balance games, often



×	Defending type																	
Attacking type	NORMAL	FIGHT	FLYING	POISON	GROUND	ROCK	BUG	GHOST	STEEL	FIRE	WATER	GRASS	ELECTR	PSYCHC	ICE	DRAGON	DARK	FAIRY
NORMAL	1×	1×	1×	1×	1×	½×	1×	0×	½×	1×	1×	1×	1×	1×	1×	1×	1×	1×
FIGHT	2×	1×	½×	½×	1×	2×	½×	0×	2×	1×	1×	1×	1×	½×	2×	1×	2×	½×
FLYING	1×	2×	1×	1×	1×	½×	2×	1×	½×	1×	1×	2×	½×	1×	1×	1×	1×	1×
POISON	1×	1×	1×	½×	½×	½×	1×	½×	0×	1×	1×	2×	1×	1×	1×	1×	1×	2×
GROUND	1×	1×	0×	2×	1×	2×	½×	1×	2×	2×	1×	½×	2×	1×	1×	1×	1×	1×
ROCK	1×	½×	2×	1×	½×	1×	2×	1×	½×	2×	1×	1×	1×	1×	2×	1×	1×	1×
BUG	1×	½×	½×	½×	1×	1×	1×	½×	½×	½×	1×	2×	1×	2×	1×	1×	2×	½×
GHOST	0×	1×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	1×
STEEL	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×	½×	1×	½×	1×	2×	1×	1×	2×
FIRE	1×	1×	1×	1×	1×	½×	2×	1×	2×	½×	½×	2×	1×	1×	2×	½×	1×	1×
WATER	1×	1×	1×	1×	2×	2×	1×	1×	1×	2×	½×	½×	1×	1×	1×	½×	1×	1×
GRASS	1×	1×	½×	½×	2×	2×	½×	1×	½×	½×	2×	½×	1×	1×	1×	½×	1×	1×
ELECTR	1×	1×	2×	1×	0×	1×	1×	1×	1×	1×	2×	½×	½×	1×	1×	½×	1×	1×
PSYCHC	1×	2×	1×	2×	1×	1×	1×	1×	½×	1×	1×	1×	1×	½×	1×	1×	0×	1×
ICE	1×	1×	2×	1×	2×	1×	1×	1×	½×	½×	½×	2×	1×	1×	½×	2×	1×	1×
DRAGON	1×	1×	1×	1×	1×	1×	1×	1×	½×	1×	1×	1×	1×	1×	1×	2×	1×	0×
DARK	1×	½×	1×	1×	1×	1×	1×	2×	1×	1×	1×	1×	1×	2×	1×	1×	½×	½×
FAIRY	1×	2×	1×	½×	1×	1×	1×	1×	½×	½×	1×	1×	1×	1×	1×	2×	2×	1×

These matchups are suitable for Generation VI.

Figure 4.1: **Top:** The Pokémon type chart. Damage done by an attack is multiplied by a factor depending on the type of the attack and the type(s) of the defending Pokémon.
Bottom: Example Pokémon statistics screen. In addition to the type multiplier, damage is also multiplied and divided by the (special) attack and defense values of the Pokémon. [11, 12] We consider the problem of setting balanced values for these statistics in a simplified version of the game.

using an evolutionary algorithm as the "outer loop" [9, 54].

Another approach, and the one we use in this chapter, is to make a game-theoretic analysis using provably optimal agents. While such analysis cannot be applied to as broad a class of games as general AI techniques, it offers lower computational costs and more precisely interpretable solution definitions.

Zero-sum games (in the game theory sense) are a natural model for games (in the entertainment sense) which pit two players against each other. In some cases the entire work may be modeled by a zero-sum game, such as in the case of Rock, Paper, Scissors—though even this simple game offers deep enough dynamics to support both human and computer tournaments [38]. Jaffe [31] investigated the effect of play restrictions on optimal agents in a two-stage zero-sum game. Even more complex works may include zero-sum components; for example, Jaffe [32] and Tavares [82] examined the metagame of choosing characters in a fighting game and choosing strategies in a RTS game respectively.

It is well-known that the Nash equilibrium of a zero-sum game can be found via linear programming [18]. Here we consider an inverse problem: given an initial zero-sum game, and a Nash equilibrium and value (i.e. expected payoff at Nash equilibrium) we would like the game to have, how can we modify the game so it has that Nash equilibrium and value? In a talk Hazard [28] gave some examples of problems of this type, but did not develop the theory further.

In this chapter, we formally define this problem in terms of *handicap functions*, which determine the payoff of the game based on the strategies picked by the two players and a *handicap variable* affecting the overall strength of each strategy. We analyze the special case where the handicap function is simply the ratio of the two

handicap variables times some initial value for that matchup, and show that such a game may be balanced using the Sinkhorn-Knopp algorithm, with its associated conditions on the existence and uniqueness of a solution. Then we move on the case where the handicap functions are general monotonic functions, and show that the solution set has properties favorable to numerical optimization. We demonstrate our algorithms on examples inspired by well-known games.

4.2 Setting

Let us now describe our formalism and how it relates to the mechanics of actual games. We consider the two-player zero-sum game, where each player chooses between a finite number of strategies. This kind of game can be represented by a matrix F, whose elements F_{ij} give the payoff if the row player picks strategy i and the column player picks strategy j. By convention, the row player is attempting to maximize this payoff, and the column player is trying to minimize it (or equivalently, maximize its negation).

Our goal is to modify this game so that it has a particular Nash equilibrium that we desire. With $n \times m$ entries in the matrix but only $n + m$ strategies, this problem is underconstrained if we are allowed to modify the entries of the matrix arbitrarily. Furthermore, the individual pairwise strategy interactions typically reflect other design goals. They may be subject to aesthetic considerations; for example, the Pokémon type chart (Figure 4.1) contains only four distinct values $0, \frac{1}{2}, 1, 2$, these being determined by intuitive relations between the types (e.g. fire burns grass, so Fire does double damage against Grass). Or the strategies may be different values of a numerical in-game statistic, with the payoff matrix being

determined by a pre-existing mathematical equation (for example, hit probability = accuracy - evasion).

Rather, it is more common to adjust the strength of each strategy rather than the payoff matrix directly. Therefore, we assign a handicap variable h_{ri} to each strategy i of the row player (the maximizer); likewise we assign a handicap h_{cj} to each strategy j of the column player (the minimizer). A high handicap indicates that the strategy is too strong at its initial state and should be made weaker. The unit cost (in e.g. money, time...) for employing a strategy is a common balancing knob; an overperforming strategy can be made more expensive, thus allowing "less" of that strategy to be employed. Alternatively, the effectiveness of the strategy could be decreased, such as by reducing the base damage for an attacker strategy, or hit points for a defender strategy.

We then define each element of the payoff matrix of the game using handicap functions whose arguments are the corresponding row and column handicaps:

$$F_{ij} = F_{ij}\left(h_{ri}, h_{cj}\right) \tag{4.1}$$

This represents how changing the handicap for the strategies i and j changes the relative advantage between the two strategies. For example, if maximizer strategy i's handicap increases, each minimizer strategy j will tend to do better against it, reducing the payoff.

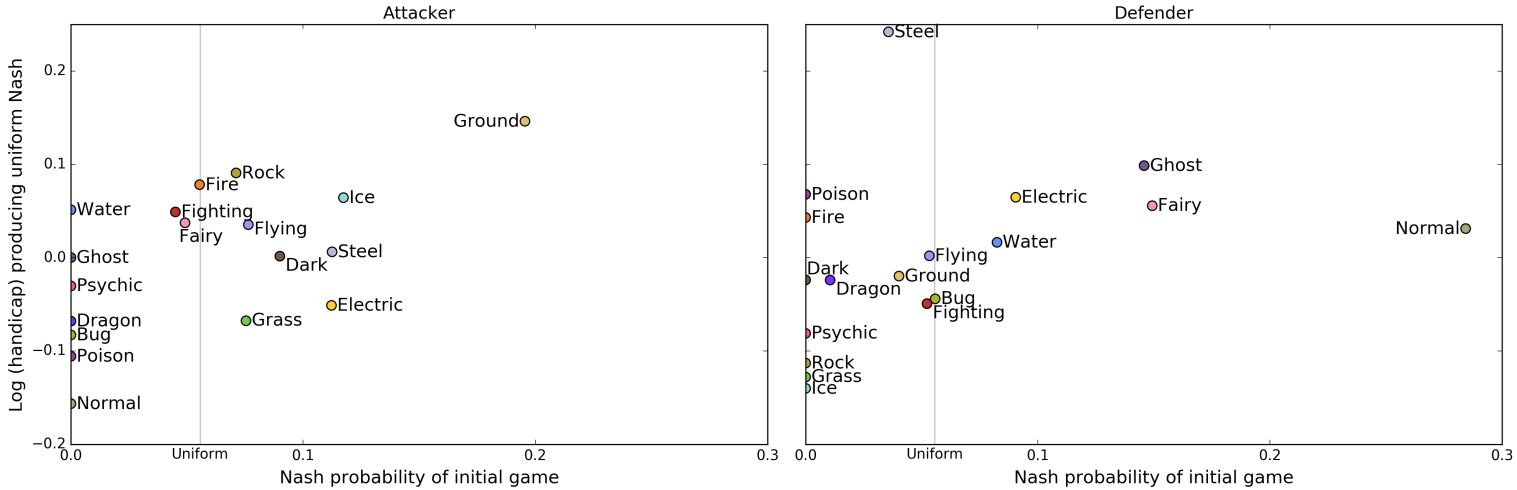


Figure 4.2: Handicaps that produce a uniform Nash equilibrium for our simplified Pokémon game, plotted against the Nash equilibrium of the original game. The attacker is the maximizer (left plot) and the defender is the minimizer (right plot). The handicap and the Nash equilibrium of the initial game are correlated, following the intuition that over-represented strategies should be weakened ("nerfed"). However, the correlation can be weak. For example, Steel defense gets a high handicap because it is strong against a wide variety of attack types; however, it is only rarely played in the initial Nash equilibrium because it is weak against the most dominant attack type, namely Ground.

Problem definition. Given...

- $n \times m$ handicap functions that define a payoff matrix as in Equation 4.1.
- A desired Nash equilibrium with strictly positive strategy weight vectors $\mathbf{w}_r$, $\mathbf{w}_c$ with one element for each of the row and column strategies respectively. As probability distributions, each of these must sum to 1.
- A desired value v of the game (i.e. expected payoff at Nash equilibrium).

...find handicap variable vectors $\mathbf{h}_r$, $\mathbf{h}_c$—again, one element for each of the row and column strategies respectively—such that the zero-sum game defined by the resulting F has that Nash equilibrium and value. Specifically, this means at the desired Nash equilibrium all strategies have expected payoff $\mathbf{p}_r, \mathbf{p}_c$ for their player equal to the desired value of the game (negated for the column player):

$$\begin{aligned} p_{ri} &= \sum_j w_{cj} F_{ij} = v && \forall i \\ p_{cj} &= -\sum_i w_{ri} F_{ij} = -v && \forall j \end{aligned} \tag{4.2}$$

We will occasionally concatenate vectors for the rows and columns into a single vector, which we will denote using unsubscripted vectors:

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_r \\ \mathbf{h}_c \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} \mathbf{p}_r \\ \mathbf{p}_c \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} \mathbf{w}_r \\ \mathbf{w}_c \end{bmatrix} \tag{4.3}$$

4.3 Multiplicative handicaps

4.3.1 Example: Pokémon types

In the *Pokémon* series of games, each attack and each defending Pokémon has an associated type. Consider a simplified version of this game where an attacker chooses the type of the attack and a defender chooses the type of the defending Pokémon, with the attacker trying to maximize the damage dealt and the defender trying to minimize it. Each type has an associated attack and defense statistic, with the damage dealt being proportional to the ratio between the attacking type's attack statistic and the defending type's defense statistic. A multiplier is applied to the damage dealt depending on the type of the attack and the type of the defending Pokémon according to the type chart shown in Figure 4.1. We wish to balance this game by setting the attack and defense statistics for each type so that the game's Nash equilibrium is the uniform distribution over each player's strategies.

The handicaps that produce this uniform Nash equilibrium are shown in Figure 4.2 and are contrasted with the Nash equilibrium of the original game. The computation took 4 ms on a single desktop computer.

4.3.2 Formalization and algorithm

Let us now formalize this example and present an algorithm for finding the solution:

- Maximizer and minimizer $\Leftrightarrow$ attacker and defender.

- Pokémon types $\Leftrightarrow$ strategies. In general there are n for the maximizer and m for the minimizer.
- Handicap $\Leftrightarrow$ inverse of attack or defense statistic (for each strategy).
- Payoff $\Leftrightarrow$ damage done; we desire that the expected damage done is v.

Finally, the handicap functions are based on an "initial" $n \times m$ nonnegative matrix A, in this example the type chart (Figure 4.1), according to:

$$F_{ij}\left(h_{ri}, h_{cj}\right) = \frac{h_{cj}}{h_{ri}} A_{ij} \tag{4.4}$$

that is, the entries of the payoff matrix are the ratio of the corresponding column and row handicaps times some initial payoff.

In this case our problem of finding the handicaps $\mathbf{h}_r, \mathbf{h}_c$ that produce a desired Nash equilibrium is equivalent to finding a scaling of the rows and columns of A such that the row and column sums, weighted by the probabilities of the desired Nash equilibrium, sum to v. Conveniently, there is an existing algorithm due to Sinkhorn and Knopp [74, 73] that does just that. It is exceedingly simple:

1. Divide each row by its (weighted) sum.
2. Divide each column by its (weighted) sum.
3. Repeat until convergence.

Sinkhorn and Knopp [74] showed that in the case where A is square and the weights are uniform, a unique solution exists if and only if A has total support; and that the algorithm converges to said solution. (A matrix has total support if every edge in the bipartite graph defined by its nonzero elements is part of a perfect

matching.) Sinkhorn [73] soon extended the algorithm to positive rectangular matrices with weights.[1]

4.3.3 Example: unit attributes

What we model as a strategy is not limited to unordered collections of items. In the context of a roleplaying, tactics, or strategy game, we can model unit attributes by considering each possible value for an attribute to be a strategy, with a better value for such an attribute to be offset by a worse value for some other attribute (e.g. cost, damage, or hit points).

For example, consider the wound roll in *Warhammer 40,000* [24]:

> **Wound Roll:** If an attack scores a hit, you will then need to roll another [6-sided] dice to see if the attack successfully wounds the target. The roll required is determined by comparing the attacking weapon's Strength characteristic with the target's Toughness characteristic, as shown on the following table:

The appropriate tables for the wound roll are shown in Figure 4.3 for the 7th and 8th editions of the game, along with the handicaps that produce a uniform Nash equilibrium and the resulting payoff matrix. The computation took 5 ms for each on a single desktop computer. Additional commentary can be found in the caption.

[1] The same problem and algorithm has found applications as diverse as computer graphics [78], web page ranking [37], and voting [77].

4.4 Monotone handicaps

While multiplicative handicaps (Equation 4.4) are applicable to many cases and admit a simple algorithm, it is quite a narrow class. In particular:

- The payoffs are unbounded with respect to handicaps, and therefore cannot represent probabilities.
- They are inherently asymmetric—there is no obvious way of representing cases where both players are choosing from the same set of strategies.

Let us therefore consider a more general class of handicap functions $F_{ij}(h_{ri}, h_{cj})$, with the goal of expressing our problem as an optimization problem, i.e. one of finding the minimum or zero of some objective function. We could then apply standard nonlinear optimization methods such as Levenberg-Marquardt; see [53] for a survey. However, merely expressing our problem as an optimization problem still leaves some major questions unanswered:

- What does the solution space look like? Could there be multiple solutions?
- Are there optimization methods likely—or better yet, guaranteed—to find a solution (if one exists)?

To produce satisfactory answers, we make the assumption that the handicap functions are *strictly monotone*:

$$\left.\begin{aligned} F_{ij}(h_{ri}, h_{cj}) &< F_{ij}(h_{ri} + x, h_{cj}) \\ F_{ij}(h_{ri}, h_{cj}) &> F_{ij}(h_{ri}, h_{cj} + x) \end{aligned}\right\} \forall \begin{array}{l} i, j, h_{ri}, h_{cj} \\ x > 0 \end{array} \tag{4.5}$$

Monotonicity is usually a reasonable assumption to make: in practical terms, it means that an increase in a strategy's handicap is always bad for the player employing it regardless of the opposing strategy.

Additionally, we will assume that $v = 0$, which we can do without loss of generality by subtracting v from all of the handicap functions F_{ij}.

4.4.1 Two-parameter monotone handicaps

We start with the more general case where the handicap functions take the two corresponding handicap variables as independent parameters. We can express our problem as a least-squares problem in terms of the expected payoff of each strategy at the desired Nash equilibrium, weighted by their probabilities at that equilibrium. Unfortunately this problem is not convex, which rules out some powerful guarantees on the efficiency of optimization methods [7]. However, we can show a weaker but still useful property. Suppose that the F_{ij} are differentiable and strictly monotone (Equation 4.5). This is enough to guarantee that the gradient of the objective function is zero if and only if it is a global minimum, a property called *invexity* [5]. Furthermore, any such point is a "perfect" solution; that is, the expected payoff of all strategies is exactly 0 at the desired Nash equilibrium. (Note that such a solution is not guaranteed to exist—for example, if all F_{ij} are bounded strictly below 0.) This at least eliminates one major failure case of non-convex optimization, that of getting stuck in a local minimum that is not a global minimum.

Theorem 1. *Let the F_{ij} be differentiable and strictly monotone functions as defined in Equation 4.5. Consider the weighted sum-of-squares error in expected payoffs of strategies at the desired Nash equilibrium*

$$z(\mathbf{h}) = \mathbf{p}(\mathbf{h}) \cdot (\mathbf{w} \odot \mathbf{p}(\mathbf{h})) \tag{4.6}$$

where $\odot$ denotes elementwise multiplication. Then $\nabla z = \mathbf{0}$ if and only if $\mathbf{p} = \mathbf{0}$.

Proof. Since the least possible value for the objective function is 0, the gradient must be zero at any point that achieves an objective of 0.

In the other direction, consider a single component of the gradient corresponding to a row strategy i, which is 0 if the overall gradient is zero:

$$\begin{aligned}(\nabla z)_{ri} &= \frac{dz}{dh_{ri}} = \frac{d}{dh_{ri}}\mathbf{p}\cdot(\mathbf{w}\odot\mathbf{p}) \\ &= 2\frac{d\mathbf{p}}{dh_{ri}}\cdot(\mathbf{w}\odot\mathbf{p}) \\ &= 2\sum_k w_{rk}p_{rk}\frac{dp_{rk}}{dh_{ri}} + 2\sum_j w_{cj}p_{cj}\frac{dp_{cj}}{dh_{ri}} \end{aligned} \tag{4.7}$$

Substituting in the derivatives of the expected payoffs (Equation 4.2), namely

$$\begin{aligned}\frac{dp_{rk}}{dh_{ri}} &= \begin{cases} \sum_j w_{cj}\frac{dF_{ri}}{dh_{ri}} & \text{for } k = i \\ 0 & \text{for } k \neq i \end{cases} \\ \frac{dp_{cj}}{dh_{ri}} &= -w_{ri}\frac{dF_{ij}}{dh_{ri}} \end{aligned} \tag{4.8}$$

we have

$$\begin{aligned}(\nabla z)_{ri} &= 2w_{ri}p_{ri}\sum_j w_{cj}\frac{dF_{ri}}{dh_{ri}} - 2\sum_j w_{cj}p_{cj}w_{ri}\frac{dF_{ij}}{dh_{ri}} \\ &= 2w_{ri}\sum_j w_{cj}\left(p_{ri}-p_{cj}\right)\frac{dF_{ij}}{dh_{ri}} \end{aligned} \tag{4.9}$$

This applies symmetrically to column strategies as well.

Now, select a strategy I with the largest absolute value of expected payoff $|p_{rI}|$. We will only explicitly treat the case that this strategy belongs to the row player, but a symmetric argument applies if it belongs to the column player. Given that p_{rI} has the largest absolute value of all expected payoffs, each $p_{rI} - p_{cj}$ either has the same sign as p_{rI} or is 0. Furthermore, given that all the weights w are strictly positive and the F_{ij} has strictly negative derivative, the only way that a term is 0 is if $p_{cj} = p_{rI}$, and the sum $(\nabla z)_{rI} = 0$ only if $p_{cj} = p_{rI}$ for all column strategies j.

But now we can use the symmetric argument in the other direction to prove that all $p_{ri} = p_{cj} = p_{rI}$ for all row strategies i. The expected payoff of the game if both players play the desired Nash equilibrium must be the same whether we sum the rows or columns first, that is

$$\sum_{i,j} w_{ri} w_{cj} F_{ij} = \sum_{i} w_{ri} p_{ri} = -\sum_{j} w_{cj} p_{cj} \tag{4.10}$$

Since all row and column weights sum to 1, we have $p_{rI} = -p_{rI}$, which means $p_{rI} = 0$, and therefore all strategies have 0 expected payoff. □

4.4.2 One-parameter monotone handicaps

Guaranteeing that all critical points are global minima is good, but we can do better if the handicap functions can be expressed as one-parameter functions of the difference between the corresponding row and column handicaps:

$$F_{ij}\left(h_{ri}, h_{cj}\right) = F_{ij}\left(h_{cj} - h_{ri}\right) \tag{4.11}$$

Another way of putting this condition is that adding any global offset to all of the handicaps does not change any of the entries of the payoff matrix. The monotone assumption (Equation 4.5) is kept by assuming these functions are strictly monotonically increasing. Note that we can express payoffs that depend on the ratio of the handicaps, such as the multiplicative handicaps of Equation 4.4, by replacing the handicap variables with their logarithms.

We can define a strictly monotone operator that is zero when we are at a solution to our problem. Finding such a zero is a special case of the variational inequality problem, which for strictly monotone operators this is guaranteed to have at most one solution (up to a global offset), and for which there exist algorithms

with guaranteed convergence; see Edwards [20] for a survey. Note that this notion of monotone *operators* is distinct from the previous notion of monotone *handicap functions*.

We begin by defining a *non*-strictly monotone operator $T : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$:

$$T(\mathbf{h}) = -\mathbf{w} \odot \mathbf{p}(\mathbf{h}) \tag{4.12}$$

This operator maps each handicap to the negative of the expected payoff of the corresponding strategy at the desired Nash equilibrium, times the weight of that strategy. Since all weights are strictly positive, this is zero if and only if all strategies have an expected payoff of 0 at the desired Nash equilibrium. Therefore our problem reduces to finding a zero of T. We now show that T is a monotone operator:

Theorem 2. *Let the handicap functions F_{ij} be strictly monotone increasing. The operator T then satisfies the monotone property*

$$(\mathbf{h}' - \mathbf{h}) \cdot (T(\mathbf{h}') - T(\mathbf{h})) \geq 0 \qquad \forall \mathbf{h}', \mathbf{h} \tag{4.13}$$

Proof. Let the prefix Δ denote the difference of a quantity from when the handicaps are $\mathbf{h}'$ minus when the handicaps are $\mathbf{h}$. Expanding the left side of Equation 4.13 yields

$$\begin{aligned}
&(\mathbf{h}' - \mathbf{h}) \cdot (T(\mathbf{h}') - T(\mathbf{h})) \\
=& -\Delta\mathbf{h}_r \cdot (\mathbf{w}_r \odot \Delta\mathbf{p}_r) - \Delta\mathbf{h}_c \cdot (\mathbf{w}_c \odot \Delta\mathbf{p}_c) \\
=& -\sum_i \Delta h_{ri} w_{ri} \sum_j w_{cj} \Delta F_{ij} \\
&+ \sum_j \Delta h_{cj} w_{cj} \sum_i w_{ri} \Delta F_{ij} \\
=& \sum_{i,j} w_{ri} w_{cj} \Delta F_{ij} \Delta (h_{cj} - h_{ri})
\end{aligned} \tag{4.14}$$

Since the F_{ij} are strictly monotone increasing ΔF_{ij} has the same sign as the change in its argument $\Delta\left(h_{cj} - h_{ri}\right)$, so the terms in this sum are always nonnegative and the sum is always nonnegative. □

Note that adding the same global offset to all handicaps does not change F, so we can fix the handicap of one strategy—say, the first row strategy—at 0 and delete it from the input of T. Likewise, by observing Equation 4.10 again, if the expected payoffs for all strategies except one are 0, the payoff of that last strategy must also be 0. Therefore towards finding a zero of T we may also delete the first row payoff from the output. If we perform this deletion, we have a strictly monotone operator:

Theorem 3. *Fix $h_{r0} = 0$, delete it from the input $\mathbf{h}$ of T, and delete the corresponding element $-w_{r0}p_{r0}$ from the output of T. The resulting operator $\bar{T} : \mathbb{R}^{n+m-1} \to \mathbb{R}^{n+m-1}$ then satisfies the strict monotone property*

$$\left(\bar{\mathbf{h}}' - \bar{\mathbf{h}}\right) \cdot \left(\bar{T}\left(\bar{\mathbf{h}}'\right) - \bar{T}\left(\bar{\mathbf{h}}\right)\right) > 0 \qquad \forall \bar{\mathbf{h}}' \neq \bar{\mathbf{h}} \tag{4.15}$$

Proof. Let overbars, as in $\bar{T}$, denote quantities with the first row strategy removed. Observe that

$$T\left(\begin{bmatrix} 0 \\ \bar{\mathbf{h}} \end{bmatrix}\right) = \begin{bmatrix} -w_{r0}p_{r0} \\ -\bar{\mathbf{w}} \odot \bar{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} -w_{r0}p_{r0} \\ \bar{T}\left(\bar{\mathbf{h}}\right) \end{bmatrix} \tag{4.16}$$

Since the first row here does not contribute to the dot product of Equation 4.13, the inequality is saturated for T if and only if it is saturated for $\bar{T}$. When does this happen? Looking at Equation 4.14 we see that, since all weights are strictly positive, each term is 0 if and only if $\Delta h_{ri} = \Delta h_{cj}$. For the sum to be 0 all

$\Delta h_{ri}, \Delta h_{cj}$ must be equal. Fixing $h_{r0} = 0$ implies that the change in that handicap is also $\Delta h_{r0} = 0$ for any $\mathbf{h}', \mathbf{h}$. Together, this means that the sum is 0 only if all $\Delta h_{ri}, \Delta h_{cj} = 0$, or in other words, $\mathbf{h}' = \mathbf{h}$.

Likewise, when we fix $h_{r0} = 0$, $\bar{\mathbf{h}}' = \bar{\mathbf{h}}$ if and only if $\mathbf{h}' = \mathbf{h}$. Putting this together, the inequality is saturated only when $\bar{\mathbf{h}}' = \bar{\mathbf{h}}$, so the inequality is strict for $\bar{\mathbf{h}}' \neq \bar{\mathbf{h}}$. □

This guarantees convergence for some existing algorithms [20]. Furthermore, the solution is unique (up to the choice of global offset), if it exists, since letting $\bar{\mathbf{h}}'$ and $\bar{\mathbf{h}}$ be two distinct solutions would contradict Equation 4.15. The global offset does not change the resulting payoff matrix, so the difference between solutions is therefore only an aesthetic choice and not a game-mechanical one. In contrast, in the two-parameter case the solution space may be more complicated, and different solutions could even have different payoff matrices, as in the following example when all handicaps are set to the same value x:

$$F = \begin{bmatrix} 2c_0 - r_0 & c_1 - 2r_0 \\ c_0 - 2r_1 & 2c_1 - r_1 \end{bmatrix} \Rightarrow \begin{bmatrix} x & -x \\ -x & x \end{bmatrix} \tag{4.17}$$

4.4.3 Symmetric games

Finally, we show that these properties apply analogously to symmetric games, where the strategies available to the two players are identical (and not merely sharing the same names as in the Pokémon example). Specifically:

- Both players have the same number of strategies ($n = m$), and the payoff matrix is square.

- $\mathbf{w}_s \triangleq \mathbf{w}_r = \mathbf{w}_c$; i.e. the desired Nash equilibrium weights are the same for both players.
- Likewise, $\mathbf{h}_s \triangleq \mathbf{h}_r = \mathbf{h}_c$; i.e. the handicaps are constrained to be the same for both players.
- $F_{ij}\left(h_{si}, h_{sj}\right) = -F_{ji}\left(h_{sj}, h_{si}\right)$; i.e. the payoff matrix F is skew-symmetric for all $i, j, \mathbf{h}_s$. Along with the weights this ensures that $\mathbf{p}_s \triangleq \mathbf{p}_r = \mathbf{p}_c$.
- $v = 0$; i.e. the desired value of the game is 0.

Theorem 4. *In the symmetric case, let $\nabla_s z$ be the derivative of the least-squares error (Equation 4.6) with respect to the shared handicaps $\mathbf{h}_s$. Then $\nabla_s z = \mathbf{0}$ if and only if $\mathbf{p} = \mathbf{0}$.*

Proof. $\nabla_s z = \nabla_r z + \nabla_c z$ where ∇_r, ∇_c are the gradients with respect to the row and column handicaps $\mathbf{h}_r, \mathbf{h}_c$ only. But by symmetry $\nabla_r z = \nabla_c z$, so $\nabla_s z = \mathbf{0}$ if and only if $\nabla_r z = \nabla_c z = \mathbf{0}$. Thus this reduces to Theorem 1. □

Theorem 5. *In the symmetric case, define the operator $T_s : \mathbb{R}^n \to \mathbb{R}^n$ as follows:*

$$T_s\left(\mathbf{h}_s\right) = -\mathbf{w}_s \odot \mathbf{p}_s\left(\mathbf{h}_s\right) \tag{4.18}$$

Fix $h_{s0} = 0$, delete it from the input $\mathbf{h}_s$ of T_s, and delete the corresponding element $-w_{s0}p_{s0}$ from the output of T_s. Then the resulting operator $\bar{T}_s : \mathbb{R}^{n-1} \to \mathbb{R}^{n-1}$ satisfies the strict monotone property of Equation 4.15.

Proof. Observe that we can input $\bar{\mathbf{h}}_s$ into the operator $\bar{T}$ of Theorem 3:

$$\bar{T}\left(\begin{bmatrix} \bar{\mathbf{h}}_s \\ 0 \\ \bar{\mathbf{h}}_s \end{bmatrix}\right) = \begin{bmatrix} -\bar{\mathbf{w}}_s \odot \bar{\mathbf{p}}_s \\ -w_{s0}p_{s0} \\ -\bar{\mathbf{w}}_s \odot \bar{\mathbf{p}}_s \end{bmatrix} = \begin{bmatrix} \bar{T}_s\left(\bar{\mathbf{h}}_s\right) \\ -w_{s0}p_{s0} \\ \bar{T}_s\left(\bar{\mathbf{h}}_s\right) \end{bmatrix} \tag{4.19}$$

The middle 0 element, resulting from fixing $h_{s0} = 0$, does not contribute to the dot product of Equation 4.15, so the dot product is simply half as much for $\bar{T}_s$ as it is for $\bar{T}$. Therefore since $\bar{T}$ is strictly monotone, so is $\bar{T}_s$. □

4.4.4 Example: matchup charts

Communities of games, particularly fighting games, often generate *matchup charts*: a matrix whose entries A_{ij} are the predicted win rate if one player picks character i and their equally-skilled opponent picks character j. Jaffe [32] investigated matchup charts and the range of possible mixed strategies that achieve some minimum win rate.

Here we analyze matchup charts in terms of handicaps. Inspired by Elo ratings, originally created to rate chess players (see [76] for recent developments), we use a logistic function as our handicap function, with the argument being the difference of the row and column handicaps plus a constant describing the specific matchup:

$$F_{ij}\left(h_{ri}, h_{cj}\right) = \frac{1}{1 + e^{-(h_{cj} - h_{ri} + \alpha_{ij})}} - \frac{1}{2} \tag{4.20}$$

where α_{ij} is chosen so that setting all handicaps to 0 reproduces the original matchup chart minus $\frac{1}{2}$, i.e. $F_{ij}\left(0, 0\right) = A_{ij} - \frac{1}{2}$. Note that this handicap function is strictly monotone, symmetric, and one-parameter. Figure 4.4 shows the result of this balancing process when applied to a *Super Street Fighter 2 Turbo* matchup chart. The computation took 4 ms on a single desktop computer.

4.5 Conclusion

Chance to wound (out of 6)

Strength \ Toughness	1	2	3	4	5	6	7	8	9	10
1	3	2	1	1	0	0	0	0	0	0
2	4	3	2	1	1	0	0	0	0	0
3	5	4	3	2	1	1	0	0	0	0
4	5	5	4	3	2	1	1	0	0	0
5	5	5	5	4	3	2	1	1	0	0
6	5	5	5	5	4	3	2	1	1	0
7	5	5	5	5	5	4	3	2	1	1
8	5	5	5	5	5	5	4	3	2	1
9	5	5	5	5	5	5	5	4	3	2
10	5	5	5	5	5	5	5	5	4	3

Strength \ Toughness	1	2	3	4	5	6	7	8	9	10
1	3	1	1	1	1	1	1	1	1	1
2	5	3	2	1	1	1	1	1	1	1
3	5	4	3	2	2	1	1	1	1	1
4	5	5	4	3	2	2	2	1	1	1
5	5	5	4	4	3	2	2	2	2	1
6	5	5	5	4	4	3	2	2	2	2
7	5	5	5	4	4	4	3	2	2	2
8	5	5	5	5	4	4	4	3	2	2
9	5	5	5	5	4	4	4	4	3	2
10	5	5	5	5	5	4	4	4	4	3

Payoff matrix and handicaps after balancing

Strength \ Toughness	1	2	3	4	5	6	7	8	9	10	SH
1	3.19	2.65	1.83	2.34	0.00	0.00	0.00	0.00	0.00	0.00	1.00
2	2.34	2.20	2.02	1.29	2.15	0.00	0.00	0.00	0.00	0.00	1.81
3	1.71	1.71	1.77	1.51	1.26	2.05	0.00	0.00	0.00	0.00	3.10
4	1.05	1.31	1.44	1.38	1.54	1.25	2.03	0.00	0.00	0.00	5.07
5	0.65	0.81	1.12	1.14	1.43	1.55	1.25	2.05	0.00	0.00	8.19
6	0.40	0.50	0.68	0.87	1.17	1.43	1.54	1.26	2.15	0.00	13.36
7	0.24	0.30	0.41	0.52	0.87	1.14	1.38	1.51	1.29	2.34	22.30
8	0.19	0.23	0.32	0.41	0.68	1.12	1.44	1.77	2.02	1.83	28.52
9	0.14	0.17	0.23	0.30	0.50	0.81	1.31	1.71	2.20	2.65	39.32
10	0.11	0.14	0.19	0.24	0.40	0.65	1.05	1.71	2.34	3.19	49.09
TH	1.00	1.25	1.72	2.20	3.68	5.99	9.68	15.82	27.10	49.09	

7th Edition

Strength \ Toughness	1	2	3	4	5	6	7	8	9	10	SH
1	1.31	0.54	0.62	0.74	0.83	0.97	1.05	1.19	1.28	1.47	1.00
2	1.74	1.28	0.98	0.59	0.66	0.77	0.83	0.95	1.02	1.17	1.26
3	1.43	1.40	1.21	0.97	1.09	0.64	0.68	0.78	0.84	0.96	1.53
4	1.14	1.40	1.28	1.15	0.87	1.01	1.09	0.62	0.67	0.76	1.92
5	0.94	1.15	1.06	1.27	1.08	0.84	0.90	1.03	1.10	0.63	2.32
6	0.81	0.99	1.14	1.09	1.23	1.08	0.77	0.88	0.95	1.08	2.71
7	0.75	0.92	1.06	1.01	1.14	1.33	1.08	0.82	0.88	1.01	2.92
8	0.68	0.84	0.96	1.15	1.04	1.21	1.30	1.11	0.80	0.91	3.21
9	0.63	0.78	0.89	1.07	0.96	1.12	1.21	1.38	1.11	0.85	3.46
10	0.57	0.70	0.81	0.97	1.09	1.02	1.09	1.25	1.34	1.15	3.82
TH	1.00	1.23	1.41	1.69	1.91	2.23	2.39	2.73	2.94	3.36	

8th Edition

Figure 4.3: Wound tables for *Warhammer 40,000*. Depending on the Strength (S) of the attacker and the Toughness (T) of the defender, a given hit has some probability of causing a wound. Tables for both 7th and 8th edition have been included. **Top:** The chance to wound out of 6 [24, 66]. **Bottom:** The resulting payoff matrix after balancing with a multiplicative handicap function along with the handicaps ("SH" for Strength and "TH" for Toughness). The payoff matrix is normalized so that the game has a value of 1; the handicaps so that a S or T of 1 has handicap 1.

Additional commentary: Like many wargames, *Warhammer 40,000* uses a points system in order to balance the strength of two opposing armies. As such the handicaps could be interpreted as a cost multiplier to assign to each Strength or Toughness value. The large number of immunities (i.e. 0 chance to wound) in the 7th edition table creates a sharp escalation of handicaps as S and T increase, as well as relatively hard counters in the resulting payoff matrix (i.e. many S-T matchups having payoff much lower or much greater than the value of the game). If desired, these could be dampened by lowering the weight of strategies that have a large number of immunities; indeed, extreme values of S and T tend to be rare in the actual game. The 8th edition table changes the probabilities from being based on the difference of S and T to being based on their ratio, and removes the immunities. This results in the handicaps increasing less sharply with S and T, as well as softer counters. More subtle changes include whether it is better to choose a slightly higher or slightly lower number than one's opponent.

Initial matchup chart

	Dhalsim	Boxer	Claw	Ryu	Chun-Li	Guile	Dee Jay	Dictator	Sagat	E. Honda	Ken	Fei-Long	Zangief	Blanka	Cammy	T. Hawk	Mean
Dhalsim	0.5	0.55	0.35	0.7	0.5	0.85	0.8	0.65	0.9	0.65	0.75	0.65	0.75	0.8	0.65	0.65	0.669
Boxer	0.45	0.5	0.55	0.6	0.45	0.6	0.7	0.65	0.8	0.65	0.65	0.7	0.55	0.75	0.75	0.55	0.619
Claw	0.65	0.45	0.5	0.6	0.6	0.7	0.6	0.5	0.75	0.6	0.6	0.65	0.65	0.65	0.6	0.7	0.613
Ryu	0.3	0.4	0.4	0.5	0.6	0.65	0.7	0.65	0.65	0.8	0.65	0.65	0.45	0.75	0.75	0.6	0.594
Chun-Li	0.5	0.55	0.4	0.4	0.5	0.6	0.65	0.7	0.65	0.75	0.65	0.4	0.75	0.7	0.6	0.55	0.584
Guile	0.15	0.4	0.3	0.35	0.4	0.5	0.55	0.55	0.65	0.8	0.65	0.75	0.8	0.6	0.8	0.65	0.556
Dee Jay	0.2	0.3	0.4	0.3	0.35	0.45	0.5	0.65	0.55	0.85	0.5	0.75	0.45	0.75	0.8	0.7	0.531
Dictator	0.35	0.35	0.5	0.35	0.3	0.45	0.35	0.5	0.65	0.35	0.4	0.65	0.45	0.35	0.8	0.7	0.469
Sagat	0.1	0.2	0.25	0.35	0.35	0.35	0.45	0.35	0.5	0.65	0.5	0.65	0.6	0.8	0.7	0.65	0.466
E. Honda	0.35	0.35	0.4	0.2	0.25	0.2	0.15	0.65	0.35	0.5	0.3	0.7	0.85	0.75	0.8	0.65	0.466
Ken	0.25	0.35	0.4	0.35	0.35	0.35	0.5	0.6	0.5	0.7	0.5	0.45	0.4	0.5	0.5	0.55	0.453
Fei-Long	0.35	0.3	0.35	0.35	0.6	0.25	0.25	0.35	0.35	0.3	0.55	0.5	0.7	0.65	0.7	0.7	0.453
Zangief	0.25	0.45	0.35	0.55	0.25	0.2	0.55	0.55	0.4	0.15	0.6	0.3	0.5	0.4	0.35	0.55	0.400
Blanka	0.2	0.25	0.35	0.25	0.3	0.4	0.25	0.65	0.2	0.25	0.5	0.35	0.6	0.5	0.4	0.7	0.384
Cammy	0.35	0.25	0.4	0.25	0.4	0.2	0.2	0.2	0.3	0.2	0.5	0.3	0.65	0.6	0.5	0.85	0.384
T. Hawk	0.35	0.45	0.3	0.4	0.45	0.35	0.3	0.3	0.35	0.35	0.45	0.3	0.45	0.3	0.15	0.5	0.359

After balancing using a logistic handicap

	Dhalsim	Boxer	Claw	Ryu	Chun-Li	Guile	Dee Jay	Dictator	Sagat	E. Honda	Ken	Fei-Long	Zangief	Blanka	Cammy	T. Hawk	Handicap
Dhalsim	0.500	0.487	0.285	0.620	0.401	0.771	0.683	0.434	0.779	0.427	0.534	0.414	0.472	0.527	0.340	0.324	0.765
Boxer	0.513	0.500	0.538	0.575	0.414	0.535	0.617	0.497	0.668	0.490	0.477	0.533	0.319	0.519	0.517	0.289	0.513
Claw	0.715	0.462	0.500	0.586	0.575	0.652	0.521	0.358	0.612	0.448	0.436	0.488	0.428	0.411	0.360	0.448	0.466
Ryu	0.380	0.425	0.414	0.500	0.589	0.612	0.642	0.523	0.509	0.696	0.503	0.503	0.259	0.545	0.543	0.356	0.408
Chun-Li	0.599	0.586	0.425	0.411	0.500	0.571	0.598	0.590	0.520	0.643	0.514	0.275	0.572	0.493	0.383	0.320	0.364
Guile	0.229	0.465	0.348	0.388	0.429	0.500	0.525	0.459	0.549	0.730	0.544	0.658	0.667	0.413	0.651	0.446	0.246
Dee Jay	0.317	0.383	0.479	0.358	0.402	0.475	0.500	0.588	0.470	0.809	0.415	0.680	0.312	0.609	0.674	0.528	0.144
Dictator	0.566	0.503	0.642	0.477	0.410	0.541	0.412	0.500	0.637	0.343	0.381	0.631	0.371	0.266	0.729	0.593	-0.118
Sagat	0.221	0.332	0.388	0.491	0.480	0.451	0.530	0.363	0.500	0.656	0.494	0.644	0.534	0.741	0.624	0.551	-0.175
E. Honda	0.573	0.510	0.552	0.304	0.357	0.270	0.191	0.657	0.344	0.500	0.290	0.689	0.808	0.676	0.735	0.544	-0.148
Ken	0.466	0.523	0.564	0.497	0.486	0.456	0.585	0.619	0.506	0.710	0.500	0.449	0.343	0.422	0.421	0.452	-0.198
Fei-Long	0.586	0.467	0.512	0.497	0.725	0.342	0.320	0.369	0.356	0.311	0.551	0.500	0.646	0.576	0.630	0.612	-0.201
Zangief	0.528	0.681	0.572	0.741	0.428	0.333	0.688	0.629	0.466	0.192	0.657	0.354	0.500	0.384	0.334	0.514	-0.445
Blanka	0.473	0.481	0.589	0.455	0.507	0.587	0.391	0.734	0.259	0.324	0.578	0.424	0.616	0.500	0.399	0.683	-0.512
Cammy	0.660	0.483	0.640	0.457	0.617	0.349	0.326	0.271	0.376	0.265	0.579	0.370	0.666	0.601	0.500	0.840	-0.516
T. Hawk	0.676	0.711	0.552	0.644	0.680	0.554	0.472	0.407	0.449	0.456	0.548	0.388	0.486	0.317	0.160	0.500	-0.590

Figure 4.4: *Super Street Fighter 2 Turbo* matchup chart before and after balancing using a logistic handicap function. The values represent the expected win rate for the row player and have been color-coded with red favoring the row player and blue favoring the column player. Data from [59]; see also commentary by Sirlin [75].

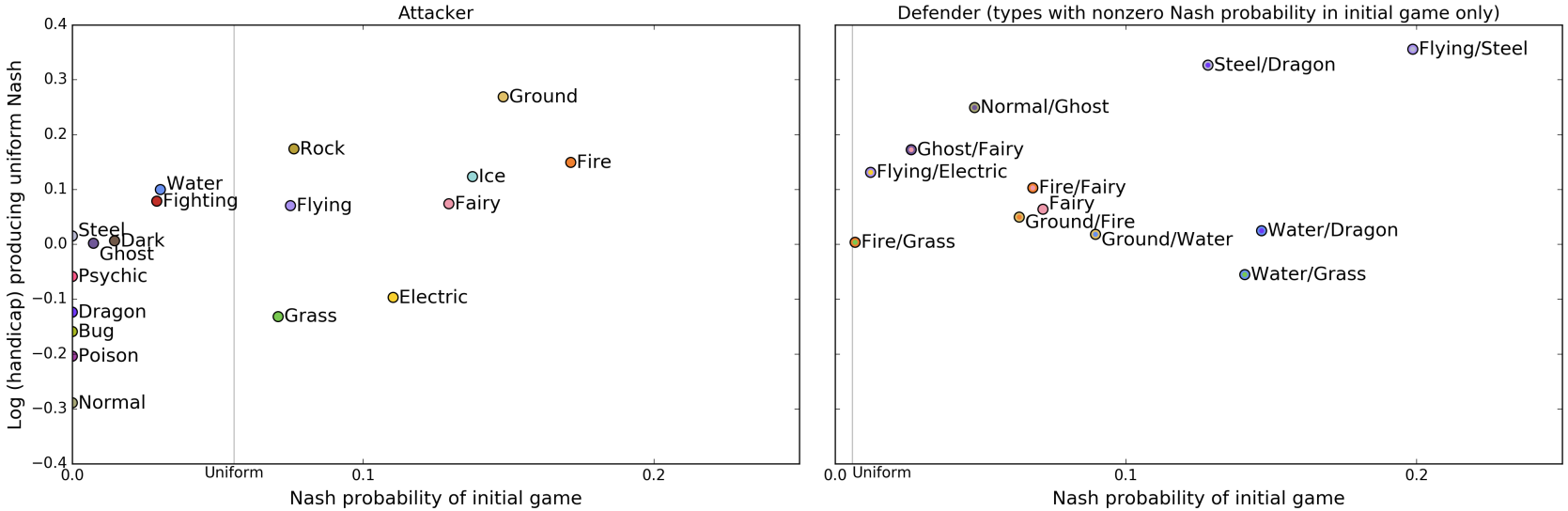


Figure 4.5: As Figure 4.2, but including dual types for the defender. To avoid excessive clutter, only defender types that have nonzero probability in the Nash equilibrium of the initial game are shown.

We have presented a formalization of balance in terms of zero-sum games and shown that it is amenable to optimization techniques under the assumption of monotonicity. Our open-source implementation, based on SciPy [35], is available at `https://github.com/ajul/zerosum`.

However, even within our setting there remain major challenges to be addressed. We discuss a couple here.

The modeling challenge. This formalization reduces the problem of balancing $n \times m$ possible pairwise matchups between strategies to modeling the individual pairwise matchups. A strength of our formulation is that we can easily slot in alternative such pairwise models, and it is thus potentially applicable to a wide variety of games. However, even evaluating two strategies against each other can be nontrivial.

For example, our fighting game matchup chart example (Figure 4.4) demonstrates that our algorithm works in the mathematical and computational sense. However, it is not clear whether the logistic handicap function itself is an accurate model, nor how to interpret the resulting handicaps quantitatively in terms of what should be changed about the characters in order to achieve a balanced game. After all, the actual win rates are determined by the fighting game proper that takes place *after* character selection, the dynamics of which are difficult to model analytically.

While perfectly accurate models may often be out of reach, approximate, learning, and/or player analytics approaches could produce good enough results to be useful. For example, learning-based approaches were used by Stanescu et al. [80, 79] in determining the outcome when two armies battle in *Starcraft*.

The combinatoric challenge. Throughout this chapter we have assumed that there is one handicap variable per strategy. This may be appropriate when all strategies are individually and explicitly specified. However, often this is not the case; rather, the strategies ultimately available to the player might be defined in a combinatorial fashion from the elements that handicaps are applied to.

In some cases balancing pairwise matchups between individual elements may still be a useful estimate; for example, in our *Warhammer 40,000* example (Figure 4.3) the result of the algorithm could provide a quick initial estimate for how to price individual units, even if does not consider entire armies. The quality of this estimate will of course depend on unmodeled factors such as unit synergies, targeting mechanics, and so forth.

In other cases this can be reasonably resolved by simply enumerating the possible combinations. For example, in the actual Pokémon games, defending Pokémon may have up to two types, not necessarily just one—Figure 4.1 shows just such an example of a Grass/Flying dual-type Pokémon. However, given that every combination of types exists in the game if and only if there is a Pokémon with that combination of types, and that each Pokémon has its own defense statistic, it is reasonable to consider every combination of types as a strategy. And while the number of strategies increases to a few hundred, the optimization is still computationally feasible (40 ms on a single desktop computer). The result is shown in Figure 4.5.

However, these are not always the case. For example, in the actual Pokémon games the player has a party of six Pokémon, and each Pokémon may have four moves (attacks) to choose from, increasing the number of strategies to an intractable level if we consider entire parties. Or for a conceptually simpler but also

intractable scenario, consider a deckbuilding game: Given n card choices and a fixed deck size k, the number of possible decks grows as $\Theta\left(n^k\right)$. This poses several problems:

- n^k is a massive number for typical n, k. It is often infeasible to even enumerate all possible decks.
- n^k is also much larger than the number of cards, which are the elements that handicaps are applied to—in other words, generally the designer can modify specific cards but not specific decks. With fewer handicap variables than decks, we cannot expect that every Nash equilibrium is possible.
- Finally, it is not even clear that having a fully mixed Nash equilibrium on decks would be desirable, even if it were achievable.

Future work might define a more reasonable objective for cases like these and find an algorithm to achieve that objective.

4.6 Appendix: one-up game

In this appendix we analyze the "one-up" game. This demonstrates the technique of analyzing a continuous strategy set by discretizing it and running it through our algorithm. We also analyze the effects of restricting one player's strategy set.

4.6.1 Introduction

The game is as follows:

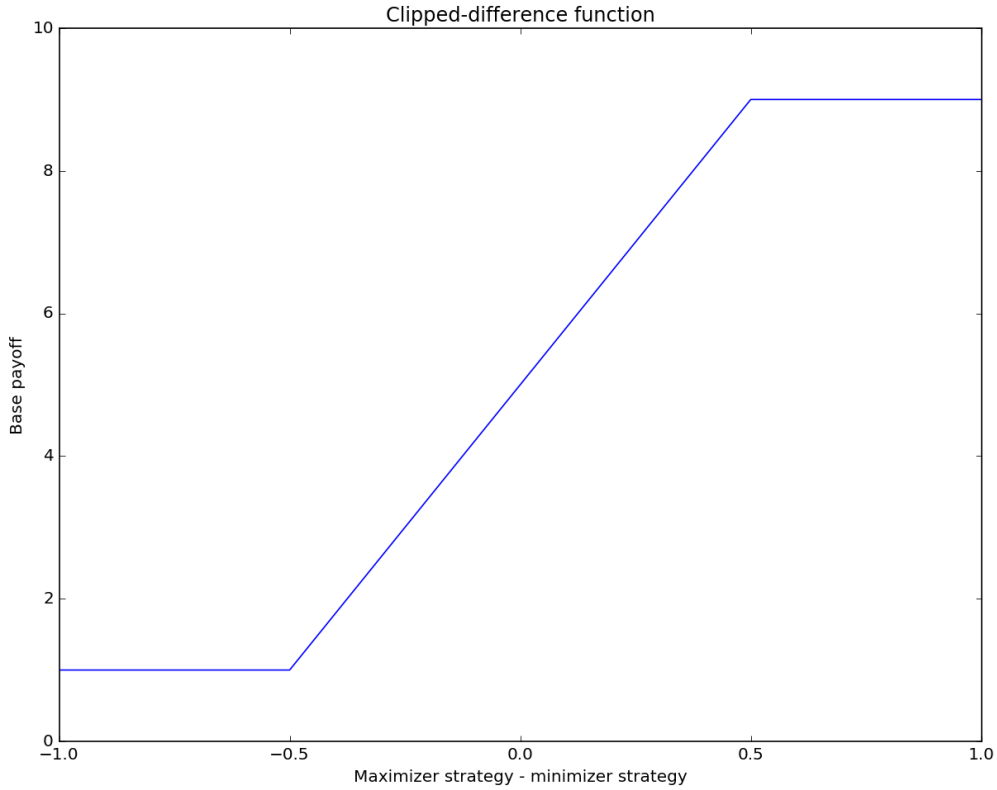


Figure 4.6: Clipped difference payoff function. As the x-axis is compressed it increasingly resembles the step function of the one-up game.

- The strategies for both players are numbers in the (continuous) interval $[0, 1)$.
- If the maximizer picks a larger number than the minimizer, they gain a base payoff of b. Otherwise their base payoff is 1.
- A uniform Nash equilibrium is to be induced by applying a multiplicative handicap.

This type of mechanic occasionally appears explicitly; for example, as in *Hearts of Iron IV*, it could represent the penetration capability of a cannon versus the thickness of armor, with a flat damage multiplier being applied if the cannon can penetrate the armor. When the base payoff is plotted as a function of the difference between the maximizer and minimizer strategies, it is a step function. As such, it

can also be seen as the limiting case of any sigmoid-shaped function as the x-axis becomes compressed; for example, "clipped-difference" payoff functions, where the probability of a successful attack is the difference of an attacker and a defender statistic, but with a minimum and maximum chance, as seen in Figure 4.6.

4.6.2 Balancing for uniform Nash equilibrium

To find the multiplicative handicaps that produce a uniform Nash equilibrium, we can use the following technique:

1. Discretize the strategy space.
2. Solve the discretized problem.
3. Graph the result.
4. Guess what function represents the result.
5. Prove that this function is indeed the solution in the continuous case.

In this case the desired multiplicative handicap for a strategy x is simply proportional to

$$h(x) = b^x \tag{4.21}$$

for both players. The payoff if the maximizer plays x and the minimizer plays y is then equal to

$$p(x, y) = b^{y-x} \cdot \begin{cases} b & \text{if } x > y \\ 1 & \text{otherwise} \end{cases} \tag{4.22}$$

$$= b^{(y-x) \bmod 1} \tag{4.23}$$

This makes every strategy symmetric, so the uniform distribution must be a Nash equilibrium.

Non-uniform Nash equilibria. For this particular game, producing a desired non-uniform Nash equilibrium is trivial: since the step function only depends on the ordering of the strategies and not their particular value, to determine the appropriate handicap for a strategy $\tilde{x}$, we can map it through the inverse cumulative distribution function of the desired Nash equilibrium, and then assign it the corresponding handicap from the uniform case:

$$x = \text{CDF}^{-1}\tilde{x} \tag{4.24}$$

$$\tilde{h}(\tilde{x}) = b^{\text{CDF}^{-1}\tilde{x}} \tag{4.25}$$

Essentially, we start with the desired non-uniform distribution and "stretch" the domain until the distribution becomes uniform. This does require that the desired Nash equilibrium is atomless so that the CDF is invertible. An atomless distribution also ensures that the result does not depend on the tiebreaking behavior, the same as with the uniform distribution.

4.6.3 Restricted strategy space

Suppose we balance the game as above, but instead of being able to play any strategy in the interval $[0, 1)$, one player has the disadvantage of only being able to play strategies in the interval $[0, a)$—perhaps they are limited by technology or economics and do not have access to higher strategies. What does this do to the Nash equilibrium and expected payoff?

Clearly the other, advantaged player has no incentive to play higher than a. Playing a already achieves the more favorable base payoff (e.g. penetrates the armor, or blocks the cannon) against all available strategies of the other player, and playing a higher strategy merely incurs a more severe handicap for no benefit. To find the exact Nash equilibrium we again use the discretization technique. Based on the result we conjecture that each player uses an extremal strategy with probability p: the player with advantage plays the "trump card" a, and the player with disadvantage "folds" and plays 0. Otherwise, with probability $1 - p$, each player plays uniformly at random in $[0, a)$. We now proceed to find p and prove that this is indeed a Nash equilibrium.

Minimizer advantage. The expected payoff if the maximizer plays x against such a strategy is

$$\underbrace{\frac{1-p}{a}\left(\underbrace{b\int_0^x b^{y-x}dy}_{\text{maximizer greater}} + \underbrace{\int_x^a b^{y-x}dy}_{\text{minimizer greater}}\right)}_{\text{minimizer plays uniformly}} + \underbrace{pb^{a-x}}_{\text{minimizer plays } a} \tag{4.26}$$

(4.27)

At Nash equilibrium this should be constant across its support. Differentiating with respect to x and setting to 0 yields

$$0 = \frac{1-p}{a}\left(b^{1-x} - b^{a-x}\right) - pb^{a-x}\ln b \tag{4.28}$$

Multiplying through by ab^x removes the dependence on x, as desired for a Nash equilibrium:

$$0 = (1-p)\left(b - b^a\right) - pab^a \ln b \tag{4.29}$$

$$p = \frac{b - b^a}{b - b^a + ab^a \ln b} \tag{4.30}$$

The expected (maximizer) payoff if the minimizer plays y against this is

$$\frac{1-p}{a}\left(\int_0^y b^{y-x}dx + b\int_y^a b^{y-x}dx\right) + pb^y \tag{4.31}$$

$$\tag{4.32}$$

Differentiating with respect to y and setting to 0 yields

$$0 = \frac{1-p}{a}\left(b^y - b^{1+y-a}\right) + pb^y \ln b \tag{4.33}$$

Multiplying through by $-ab^{a-y}$ gives

$$0 = (1-p)\left(-b^a + b\right) - pab^a \ln b \tag{4.34}$$

exactly the same.

Substituting this back in, we have expected payoff

$$\frac{b^a\,(b-1)}{b-b^a+ab^a\ln b} \tag{4.35}$$

Maximizer advantage. This is the same except the final term changes to pbb^{-x} and pbb^{y-a} for the maximizer and minimizer respectively, resulting in

$$p=\frac{b-b^a}{b-b^a+ab\ln b} \tag{4.36}$$

and expected payoff

$$\frac{b\,(b-1)}{b-b^a+ab\ln b} \tag{4.37}$$

Commentary. These are plotted for $b=9$ in Figures 4.7 and 4.8, with further commentary in the captions. (Imagine e.g. a jump from a 10% hit chance to a 90% hit chance.)

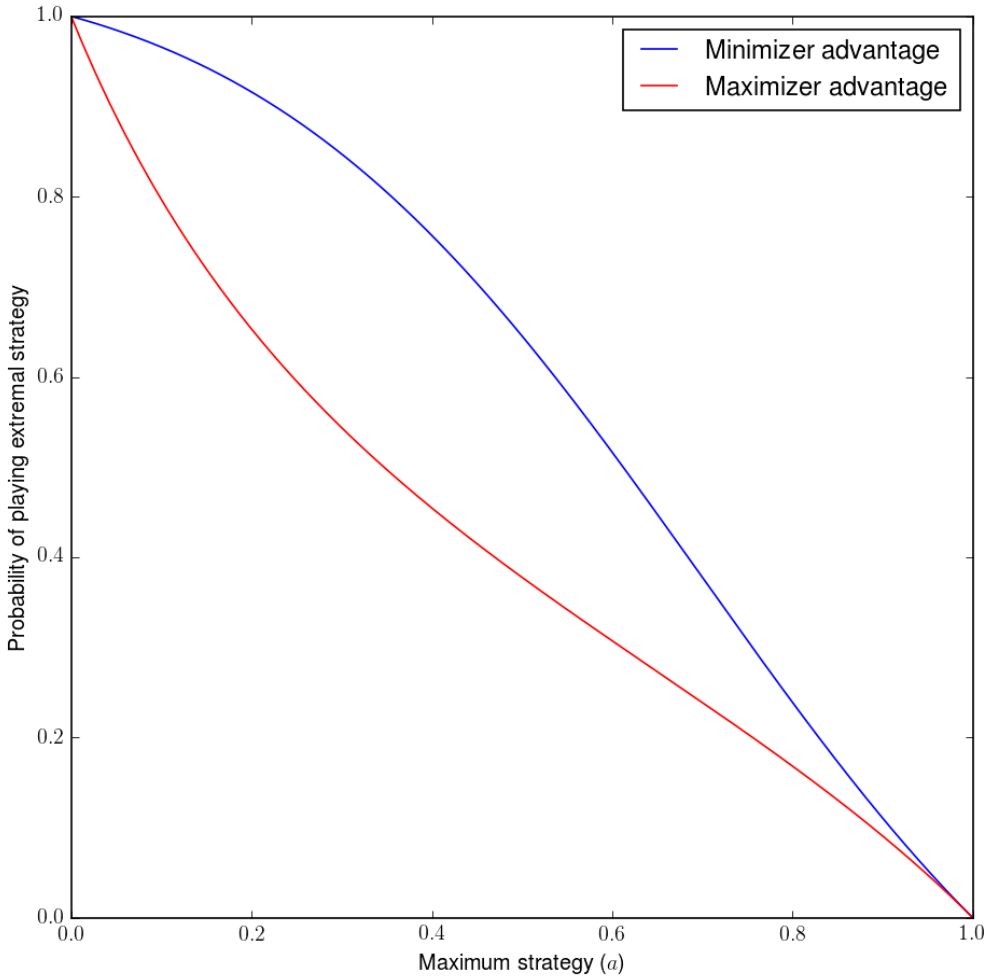


Figure 4.7: Probability at Nash equilibrium of playing an extremal strategy, i.e. the maximum strategy a for the player with advantage, or the minimum strategy 0 for the player with disadvantage. This plot is for maximum payoff $b = 9$.
For $a \neq 0, 1$ the probability of extremal strategy is not symmetric with respect to which player has the advantage. Namely, for a given a, extremal strategies are played more often when the minimizer has the advantage than when the maximizer has the advantage. This gap increases with b. In either case the Nash equilibrium becomes more uniform (i.e. p decreases) as a increases.

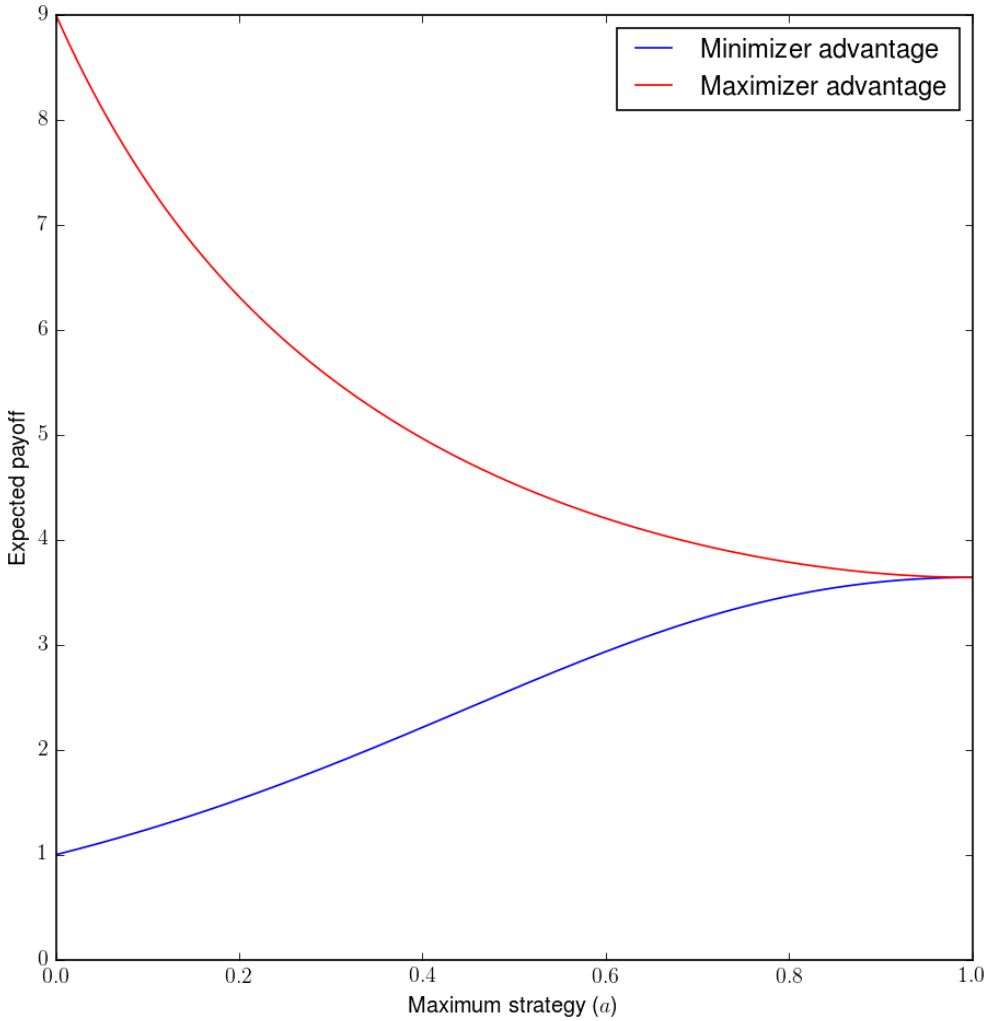


Figure 4.8: Expected payoff at Nash equilibrium in the one-up game. This plot is again for maximum payoff $b = 9$.
Having the advantage matters less as a increases, as shown by the difference in expected payoffs for a given a depending on whether the minimizer or the maximizer has the advantage.

CHAPTER 5

CONCLUSION

5.1 Use in practice

"Caliber: camera localization and calibration using rigidity constraints". CALIBER fulfilled its original purpose of calibrating our spherical gantry under several different configurations, participating in the data-gathering process for several papers from our lab [34, 63, 36].

"Simulating the structure and texture of solid wood". Autodesk (who had funded this work) re-implemented my wood texture in Fusion 360, which was a great aid to producing renderings for the publication. The wood texture is a part of that product today.

"Balancing zero-sum games with one variable per strategy". Published only a few months before I wrote this thesis, it is too soon for this work to have had much use in practice. I hope that it, or extensions thereof, will have such an impact in the future.

5.2 Future research

Combinatorial optimization. Consider these possible extensions to the works in this dissertation:

- Caliber: Somehow automatically determine the structure of the calibration setup rather than have the user define it directly.
- Game balance: When strategies are defined in a combinatorial manner, the number of strategies may far exceed the number of balancing variables. In most cases one would expect the Nash equilibrium to support a number of strategies at most roughly equal to the number of balancing variables.

Both of these problems add a combinatorial aspect to the balancing problem—in addition to the continuous optimization aspect, one must also make discrete choices, namely what tree to use in the case of Caliber, and which strategies to support in the case of balancing games. A major challenge in this direction is that combinatorial problems can often be intractable, as demonstrated in our proof of the NP-hardness of the SL-R problem. Discrete choices rob us of the powerful tools of calculus, and when choices are combinatorially defined there may be a massive number of them to examine. Several techniques, such as branch-and-bound and Monte Carlo Tree Search [8, 94], have been used to attack combinatorial optimization problems; a possible direction of research would be to see how these might be applied in our particular cases.

Incremental systems. Systems are not always static; rather, components may be added over time. Again, consider some possible extensions to the works in the previous chapters:

- Caliber: Given an existing set of measurements and sensors, algorithmically recommend additional measurements and/or sensors to add to the system that might improve the quality of the calibration.

- Game balance: In many games, not all strategies are available to the players at the beginning of the game, but rather must be unlocked using some process (e.g. money, skill points, technology). How do we define and solve game balance in this type of case? The analysis of the one-up game (Section 4.6) is one first step towards this.

BIBLIOGRAPHY

[1] Erik Andersen, Yun-En Liu, Rich Snider, Roy Szeto, and Zoran Popović. Placing a value on aesthetics in online casual games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1275–1278. ACM, 2011.

[2] Erik Andersen, Yun-En Liu, Richard Snider, Roy Szeto, Seth Cooper, and Zoran Popovic. On the harmfulness of secondary game objectives. In *Proceedings of the Sixth International Conference on the Foundations of Digital Games*, 2011.

[3] Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Richard Snider, Jeff Lowdermilk, David Truong, Seth Cooper, and Zoran Popovic. The impact of tutorials on games of varying complexity. In *Proceedings of the 2012 annual conference on Human factors in computing systems*, 2012.

[4] Harold Oliver Beals and Terry Chaffin Davis. Figure in wood: an illustrated review. Technical Report 486, Alabama Agricultural Experiment Station, Auburn, Alabama, USA, January 1977.

[5] A Ben-Israel and B Mond. What is invexity? *The ANZIAM Journal*, 28(1):1–9, 1986.

[6] Jean-Yves Bouguet. Camera calibration toolbox for Matlab, July 2010.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[8] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[9] Cameron Bolitho Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

[10] John W Buchanan. Simulating wood using a voxel approach. In *Computer Graphics Forum*, volume 17, pages 105–112. Wiley Online Library, 1998.

[11] Bulbapedia. Statistic — bulbapedia, the community driven pokémon encyclopedia. `https://bulbapedia.bulbagarden.net/wiki/Statistic`, 2017. [Online; accessed 3-July-2017].

[12] Bulbapedia. Type — bulbapedia, the community driven pokémon encyclopedia. `http://bulbapedia.bulbagarden.net/wiki/Type`, 2017. [Online; accessed 11-April-2017].

[13] Andy Chambers et al. *Chapter Approved*. Games Workshop Ltd., 2001.

[14] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[15] H.H. Chen. A screw motion approach to uniqueness analysis of head-eye geometry. In *CVPR*, pages 145–151, 1991.

[16] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 287–294, New York, NY, USA, 2003. ACM.

[17] Robert L. Cook and Tony DeRose. Wavelet noise. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 803–811, New York, NY, USA, 2005. ACM.

[18] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[19] Yoann Dieudonné, Ouiddad Labbani-Igbida, and Franck Petit. Deterministic robot-network localization is hard. *Trans. Rob.*, 26(2):331–339, April 2010.

[20] Mclean R Edwards. Five classes of monotone linear relations and operators. In *Computational and Analytical Mathematics*, pages 375–400. Springer, 2013.

[21] Sandro Esquivel, Felix Woelk, and Reinhard Koch. Calibration of a multi-camera rig from non-overlapping views. In Fred Hamprecht, Christoph Schnörr, and Bernd Jähne, editors, *Pattern Recognition*, volume 4713 of *Lecture Notes in Computer Science*, pages 82–91. Springer Berlin / Heidelberg, 2007.

[22] Carlos Estrada, José Neira, and Juan D. Tardós. Finding good cycle constraints for large scale multi-robot slam. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 2427–2431, Piscataway, NJ, USA, 2009. IEEE Press.

[23] Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. Gabor noise by example. *ACM Trans. Graph.*, 31(4):73:1–73:9, July 2012.

[24] Games Workshop. Warhammer 40,000 battle primer. `https://www.games-workshop.com/en-US/Warhammer-40000-Rules`, 2017. [Online; accessed 21-July-2017].

[25] Venu Madhav Govindu. Combining two-view constraints for motion estimation. In *In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 218–225, 2001.

[26] V.M. Govindu. Lie-algebraic averaging for globally consistent motion estimation. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–684–I–691 Vol.1, 2004.

[27] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2004.

[28] Christopher J. Hazard. What every game designer should know about game theory. Triangle Game Conference, April 2010. Raleigh, North Carolina.

[29] Greg Heath. How to solve the matrix equation xax=b, March 2008.

[30] R. Bruce Hoadley. *Understanding wood: a craftsman's guide to wood technology*. Taunton Press, Newtown, Connecticut, USA, 1980.

[31] Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. In *Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012.

[32] Alexander Benjamin Jaffe. *Understanding game balance with quantitative methods*. PhD thesis, University of Washington, 2013. `https://digital.lib.washington.edu/researchworks/handle/1773/22797`.

[33] Wenzel Jakob. Mitsuba renderer. http://www.mitsuba-renderer.org, 2010.

[34] Wenzel Jakob, Eugene d'Eon, Otto Jakob, and Steve Marschner. A comprehensive framework for rendering layered materials. *ACM Transactions on Graphics (ToG)*, 33(4):118, 2014.

[35] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

[36] Pramook Khungurn and Steve Marschner. Azimuthal scattering from elliptical hair fibers. *ACM Transactions on Graphics (TOG)*, 36(2):13, 2017.

[37] Philip A. Knight. The sinkhorn–knopp algorithm: convergence and applications. *SIAM Journal on Matrix Analysis and Applications*, 30(1):261–275, 2008.

[38] Byron Knoll, Daniel Lu, and Jonathan Burdge. Rock paper scissors programming competition. `http://www.rpscontest.com/`, 2017. [Online; accessed 5-April-2017].

[39] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[40] R.K. Kumar, A. Ilie, J.-M. Frahm, and M. Pollefeys. Simple calibration of non-overlapping cameras with a mirror. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7, 2008.

[41] Rainer Kümmerle, Giorgio Grisetti, and Wolfram Burgard. Simultaneous calibration, localization, and mapping. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3716–3721. IEEE, 2011.

[42] Ares Lagae and Philip Dutré. A procedural object distribution function. *ACM Trans. Graph.*, 24(4):1442–1461, October 2005.

[43] Ares Lagae, Craig S. Kaplan, Chi-Wing Fu, Victor Ostromoukhov, and Oliver Deussen. Tile-based methods for interactive applications. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 93:1–93:267, New York, NY, USA, 2008. ACM.

[44] Ares Lagae, Sylvain Lefebvre, Rob Cook, Tony DeRose, George Drettakis, David S Ebert, JP Lewis, Ken Perlin, and Matthias Zwicker. A survey of procedural noise functions. In *Computer Graphics Forum*, volume 29, pages 2579–2600. Wiley Online Library, 2010.

[45] Ares Lagae, Sylvain Lefebvre, George Drettakis, and Philip Dutré. Procedural noise using sparse gabor convolution. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 54:1–54:10, New York, NY, USA, 2009. ACM.

[46] Ares Lagae, Sylvain Lefebvre, and Philip Dutré. Improving gabor noise. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1096–1107, 2011.

[47] Laurent Lefebvre and Pierre Poulin. Analysis and synthesis of structural textures. In *Graphics Interface*, volume 2000, pages 77–86, 2000.

[48] Sylvain Lefebvre. Part iv: Runtime texture synthesis. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[49] Sylvain Lefebvre. *Synthése de textures par l'exemple pour les applications interactives*. Habilitation thesis, Université de Lorraine, jun 2014.

[50] John-Peter Lewis. Texture synthesis for digital painting. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 245–252. ACM, 1984.

[51] John-Peter Lewis. Algorithms for solid noise synthesis. *ACM SIGGRAPH Computer Graphics*, 23(3):263–270, 1989.

[52] Derek Lomas, Kishan Patel, Jodi L Forlizzi, and Kenneth R Koedinger. Optimizing challenge in an educational game using large-scale design experiments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 89–98. ACM, 2013.

[53] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. Methods for non-linear least squares problems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2004.

[54] Tobias Mahlmann, Julian Togelius, and Georgios N Yannakakis. Evolving card sets towards balancing dominion. In *2012 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2012.

[55] Stephen R. Marschner, Stephen H. Westin, Adam Arbree, and Jonathan T. Moon. Measuring and modeling the appearance of finished wood. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 727–734, New York, NY, USA, 2005. ACM.

[56] Stephen J. Maybank and Oliver D. Faugeras. A theory of self-calibration of a moving camera. *IJCV*, 8(2):123–151, 1992.

[57] Morgan McGuire. Computer graphics archive, August 2011.

[58] D. Nister, F. Kahl, and H. Stewenius. Structure from motion with missing data is np-hard. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–7, 2007.

[59] nohoho. Super turbo - new arcadia diagram. `http://curryallergy.blogspot.com/2008/11/super-turbo-new-arcadia-diagram.html`, November 2008. [Online; accessed 28-April-2017].

[60] Alexis John Panshin and Carl De Zeeuw. *Textbook of wood technology*. McGraw-Hillm Inc., New York, New York, USA, 3rd edition, 1970.

[61] F.C. Park and B.J. Martin. Robot sensor calibration: solving $AX = XB$ on the euclidean group. *IEEE Trans. on Robotics and Automation*, 10(5):717 –721, oct 1994.

[62] Darwyn R. Peachey. Solid texturing of complex surfaces. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 279–286, New York, NY, USA, 1985. ACM.

[63] Huaishu Peng, Rundong Wu, Steve Marschner, and François Guimbretière. On-the-fly print: Incremental printing while modelling. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 887–896. ACM, 2016.

[64] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, pages 287–296, New York, NY, USA, 1985. ACM.

[65] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. ACM.

[66] James Rodriguez. New community made 8th edition 40k wound chart. `https://spikeybits.com/2017/05`, May 2017. [Online; accessed 21-July-2017].

[67] A. Rollings and D. Morris. *Game Architecture and Design*. Coriolis, 2000.

[68] Damien Sellier and Jonathan J Harrington. Phloem transport in trees: A generic surface model. *Ecological Modelling*, 290:102–109, 2014.

[69] Damien Sellier, Michael J Plank, and Jonathan J Harrington. A mathematical

framework for modelling cambial surface evolution using a level set method. *Annals of botany*, 108(6):1001–1011, 2011.

[70] Peter Shirley and Steve Marschner. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 3rd edition, 2009.

[71] Yiu Cheung Shiu and Shaheen Ahmad. Calibration of wrist-mounted robotic sensors by solving homogeneoustransform equations of the form $AX = XB$. *IEEE Trans. on Robotics and Automation*, Jan 1989.

[72] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[73] Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[74] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[75] David Sirlin. Game balance and yomi. `http://www.sirlin.net/articles/game-balance-and-yomi`, 2014. [Online; accessed 16-May-2017].

[76] Yannis Sismanis. How i won the "chess ratings-elo vs the rest of the world" competition. *arXiv preprint arXiv:1012.4571*, 2010.

[77] Warren D Smith. Sinkhorn ratings, and new strongly polynomial time algorithms for sinkhorn balancing, perron eigenvectors, and markov chains. July 2005. `http://scorevoting.net/WarrenSmithPages/homepage/sinkhornv.pdf`.

[78] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4):66, 2015.

[79] Marius Stanescu, Nicolas Barriga, and Michael Buro. Using lanchester attrition laws for combat prediction in starcraft. In *Eleventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2015.

[80] Marius Stanescu, Sergio Poo Hernandez, Graham Erickson, Russel Greiner, and Michael Buro. Predicting army combat outcomes in starcraft. In *Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*. Citeseer, 2013.

[81] Klaus H. Strobl and Gerd Hirzinger. Optimal hand-eye calibration. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4647–4653, 2006.

[82] Anderson Tavares, Hector Azpúrua, Amanda Santos, and Luiz Chaimowicz. Rock, paper, starcraft: Strategy selection in real-time strategy games. In *Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2016.

[83] Olivier Terraz, Guillaume Guimberteau, Stéphane Mérillou, Dimitri Plemenos, and Djamchid Ghazanfarpour. 3gmap l-systems: an application to the modelling of wood. *The Visual Computer*, 25(2):165–180, 2009.

[84] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research*, 25(5/6):403–430, 2005.

[85] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Trans. on Robotics and Automation*, 3(4):323–344, 1987.

[86] R.Y. Tsai and R.K. Lenz. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Trans. on Robotics and Automation*, 5(3):345 –358, jun 1989.

[87] Jarke J. van Wijk. Spot noise texture synthesis for data visualization. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 309–318, New York, NY, USA, 1991. ACM.

[88] Tobi Vollebregt et al. Spring rts engine. `https://springrts.com/`, 2017. [Online; accessed 10-November-2017].

[89] Gregory J Ward. Measuring and modeling anisotropic reflection. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 265–272. ACM, 1992.

[90] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, Greg Turk, et al. State of the art

in example-based texture synthesis. In *Eurographics 2009, State of the Art Report, EG-STAR*, pages 93–117, 2009.

[91] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 291–294, New York, NY, USA, 1996. ACM.

[92] Christopher Zach, Manfred Klopschitz, and Manfred Pollefeys. Disambiguating visual relations using loop constraints. In *CVPR*, pages 1426–1433. IEEE, 2010.

[93] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *ICCV*, pages 666–673, 1999.

[94] Alexander Zook, Brent Harrison, and Mark O Riedl. Monte-carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th Conference on the Foundations of Digital Games*, 2015.